

SEMANTIC SEGMENTATION OF MICROSCOPIC BLOOD
IMAGE DATA USING SELF-TRAINING TO AUGMENT SMALL
TRAINING SETS AND ITS APPLICATION FOR COUNTING
CELLS

by

Junliang Luo

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2019

© Copyright by Junliang Luo, 2019

Table of Contents

List of Tables	iv
List of Figures	vi
Abstract	viii
Acknowledgements	ix
Chapter 1 Introduction	1
1.1 Problem and task	1
1.2 Overview	2
1.3 Thesis outline	4
Chapter 2 Basic concepts	5
2.1 Computer vision task	5
2.1.1 Semantic segmentation	7
2.2 Machine learning	8
2.2.1 Supervised learning	8
2.2.2 Semi-supervised learning	8
2.3 Neural Networks	8
2.3.1 Convolutional Neural Networks	12
2.3.2 Neural Networks for semantic segmentation	13
2.4 Evaluation measures for semantic segmentation	15
2.4.1 Region based measure	15
2.4.2 Contour based measure	17
2.4.3 Benchmark dataset for semantic segmentation	18
2.5 Limited data problem	20
2.5.1 Labeling data	22
2.5.2 Using outside data resources	23
Chapter 3 Breakdown Experiment	25
3.1 Motivation	25
3.2 Related work	27

3.3	Methods	27
3.4	Results	31
Chapter 4	Approaches for semantic segmentation from limited labeled microscopic blood image dataset	41
4.1	Motivation	41
4.2	Related Work	41
4.3	Self-training (Bootstrapping) method	43
4.3.1	Results on semantic segmentation	45
4.4	Counting algorithm	49
4.5	Discussion	53
4.5.1	Active learning	54
Chapter 5	Conclusion	56
5.1	Summary of results	56
5.2	Future work	57
	Appendices	59
Appendix A	Dataset sizes in papers of deep learning models for semantic segmentation	60
	Bibliography	62

List of Tables

2.1	The growing dataset size of Pascal VOC and MS COCO benchmarks	20
2.2	Summary of the required label types and cost estimation for computer vision tasks	23
2.3	Domains of applications and the related datasets	23
3.1	Five classes of the synthetic 2D-Gaussian dataset, where c_{big} , c_{small} , c_{long} are constants for thresholding when generating the Gaussian objects	29
3.2	Examples of the training data in descending order of training size in configurations: [number of images - resolution - number of instances]	30
3.3	The number of Gaussian objects and their class distribution in training samples in format[class 0: Normal class 1: Big class 2: Small class 3: Long]	31
3.4	Example of the inference results of a testing image of the model trained on 1/25 of a full image (resolution 50x50) containing only 2 cells.	33
3.5	Example of the inference results of testing images for training sets group of [1 image], by models trained on in descending order of training size in configurations: [number of images - [part size - resolution] - number of instances]. In overlays, blue refers to correctly classified pixels, yellow refers to pixels of detected, type mis-classified Gaussian objects, and red refers to pixels of undetected Gaussian objects	37
3.6	Example of the inferred results of testing images for training sets group of [3 images]. The format of the configuration and the overlays is as same as those of [1 image] above	38
3.7	Example of the inferred results of testing images for training sets group of [8 images]. The format of configuration and overlay are as same as those of [1 image] above	39

4.1	Summary of mean and variance of segmentation accuracies, IoU, PPV, TP, TN, P over 5 runs. Left: Performance measures in evaluation after training on labeled data (before self-training). Right: After self-training.	47
4.2	Evaluation of cell counting on 50 testing images by three pairs of (c1,c2), iterations of self-training=7, n=5. The numbers in columns: P(positive), GT(ground truth), TP(true positive), FP(false positive), FN(false negative) correspond to mean and standard deviation of the number of cells, in columns: TPR(true positive rate), PPV(positive predictive value) correspond to rates calculated from the whole 50 samples.	51
4.3	Evaluation of counting on 10 testing images of synthetic 2D-Gaussian by two pairs of (c1,c2), iterations of self-training=8, n=6.	52
A.1	Summary of the dataset and the number of data used for influential deep learning semantic segmentation networks. Samples are denoted as T : training, E: Evaluation(validation or testing).	61

List of Figures

2.1	Example of computer vision tasks including object localization, object detection, instance segmentation, and semantic segmentation. (The raw image is from [74]).	6
2.2	An MLP with one hidden layer. x denotes the value of input nodes, a denotes the values before activation function F of hidden nodes, and y denotes the value of output nodes.	11
2.3	Example of a basic structure of a encoder-decoder network for semantic segmentation. (W , H denote the weight and height of the input. C denotes the number of classes).	13
2.4	Encoder-decoder network with shortcut connections	14
2.5	Maxpooling indices are stored during the feature extracting	14
2.6	The architecture of the U-Net. Entire feature maps are copied and concatenated from encoder to decoder layers	15
2.7	Simple bounding box example of IoU and F1 measures: ratios of statistical measures.	17
3.1	Example image and ground truth annotation of BBBC cells data	28
3.2	Example image and ground truth annotation of synthetic 2D Gaussians dataset (100 Gaussian objects)	28
3.3	The results of training and testing accuracy of mean IoUs given the training sample size and how many objects in training images, tested on 10 testing images (950×950), averaged over 6 runs. Variances are shown as errorbars	32
3.4	Testing accuracies with the regression line of proposal relationship: accuracy = $1 - (bn^{-1/2})$, and accuracy = $1 - (a + bn^{-2})$	32
3.5	Testing results for mean IoUs given the training sample size in configurations: [number of images - [fraction part size] - number of instances], averaged over 6 runs. Variances are shown as shaded	34
3.6	Training and testing results for mean IoUs given the training sample size and how many objects in training images, averaged over 6 runs. Variances are shown by the error bars	35

3.7	Testing results for [1 image], [3 images] [8 images] from top to bottom, showing IoUs of each class given the training sample sizes and how many objects in training images, averaged over 6 runs. Variances are shown as shaded	36
4.1	The work-flow of our self-training model based on U-Net . . .	45
4.2	Average testing accuracies and IoUs for training the network with different initial training sizes in configurations given by: [percent of a full image, pixels, number of objects]	46
4.3	Inference results: (a0)x, (a0)y: Input data and label. (a1)-(a4): After training on labeled data. (b1): After 15 self-training iterations(ends up with training on 1 labeled data + 15 pseudo-labeled data) using mean probability pseudo-labels. (b2): After 28 self-training iterations using mean binary threshold (0.5) pseudo-labels. (b3): After 28 self-training iterations using mean probability threshold (0.5) pseudo-labels. (b4): After 15 self-training iterations using nearby pixels threshold (8) pseudo-labels.	48
4.4	A test sample for the counting process on cells images: (a): Prediction after training on 1 labeled data. (b): Prediction after 7 self-training iterations (1 labeled data + 7 pseudo-labeled data. Pseudo label: Nearby pixels threshold). (c): Laplacian filtered(b). (d): <i>map1</i> in Alg 1. (e): <i>map2</i> in Alg 1. (f): <i>diff</i> in Alg 1. (g): Dots prediction	51
4.5	A test sample for the counting on 2D-Gaussian synthetic images. Left: predictions. Right: Ground truth (centres of each 2D-Gaussian objects). P:192, TP:183, FP:9, FN:17.	53
4.6	A self-made dot annotation labeling tool	55

Abstract

Semantic segmentation is a computer vision task of assigning a label describing the content to each pixel in an image. There has been a lot of progress in this area using deep neural networks with an encoder-decoder structure. However, these methods usual place reliance on learning from a lot of data. In this thesis, we study the performance of semantic segmentation networks trained on small labeled training sets. This is thereby studied in the context of the application of detecting and counting red blood cells in microscopic images of a recently developed lensless microscope. In addition, we use a specifically designed generic dataset to investigate performance more systematically.

We first study the performance breakdown with the sizes of the training sets using a synthetic 2D-Gaussian dataset. Then for our microscopic blood images, we evaluate a method similar to an Expectation-Maximization approach to improve performance with limited labeled training data through a self-training procedure. In this self-training procedure, we add unlabeled data to the training set using the model's own prediction as pseudo-labels for the unlabeled data. We compare several methods of producing pseudo-labels and show that only one of them improved lightly the segmentation performance. Indeed, most of the methods lead to a deterioration of the accuracy. However, we also noticed that these pseudo-labels that lowered the IoU accuracy lead to a rapid intensity change in the per-pixel prediction map at locations associated with edges. Based on this finding we propose a new counting algorithm and show that this method results in a testing error rate of 6-9% on counting red blood cells.

Acknowledgements

I would like to thank my parents for supporting me in pursuing graduate studies. I would like to thank my supervisor Dr. Thomas Trappenberg for his immense knowledge and passion on research. With his guidance, I feel lucky to have the chance to work on the research for the first time. I would like to thank Dr. Sageev Oore for helping me a lot for my research and study, and also for being the reader. Besides my advisors, I would like to thank Dr. Malcolm Heywood for being the reader. Thank you also Yoshi for giving me some small decorations on my table.

Chapter 1

Introduction

Computer vision is a popular interdisciplinary field since the late 1960s, to build visual systems for machines [69]. In recent years, machine learning made considerable progress in various computer vision tasks [60, 37, 70, 76]. In particular, the specific set of machine learning algorithms called deep neural networks, exhibit good generalization on image data and has become a strong tool for solving application targets related to computer vision, including autonomous driving [12, 31], a search engine for images [71], face recognition [63], playing games (classic video games and the famous game of Go) [26, 66], object tracking in video [71], etc.

Semantic segmentation is one of the core computer vision problems that drive the development of scene understanding. The task is to assign each pixel of an image a class label. Deep networks tackling the semantic segmentation problem, surpasses traditional CV algorithms in terms of the accuracy.

Despite the impressive performance of deep networks on semantic segmentation, the deep network is tightly coupled with a large amount of data and labels, since there are a huge number of parameters in a deep network. For example, the well known 16 layers VGG convolution network [67] has over 138 million parameters.

1.1 Problem and task

There is no guarantee for the amount of labeled data in a real world application. The situation exists that the novel deep networks are applied to a dataset in a limited small size.

The project we worked on is to apply a deep network to segment human blood cells in microscopic images from a recently developed lensless microscope. The goal for the project is to use the model to segment cells for further analysis. For example, the model can be used to segment out a specific rare type of blood cells, which human cannot distinguish from other types very well. The segmentation results can also be

used to count the number of a specific type of recognized cells.

Here our focused discussion is how much labeled data is necessary to get an expected segmentation performance for our application. Since we do not have accurate annotations for this microscopic dataset, we look into how many images we should label. Also, if we can only have a small set of images labeled, we need to propose methods or model structures that can do semantic segmentation on this microscopic cell dataset from a small set of labeled training images.

Generally, there is no simple conclusion of how much training data is required to achieve the desired performance for deep networks. Therefore, it is worthwhile for performing the experiment of training a deep network on a small labeled dataset, and do analysis on how the accuracy varies with different training set sizes.

We find that a small dataset may be sufficient to learn enough useful representation for deep networks in some cases. For example, the U-Net [57] achieves a 6% pixel error by being trained on only 30 images of ventral nerve cord(VNC) medical dataset [9]. In the paper of Siamese Neural Networks [35], 70% classification predictions are correct on MNIST dataset [40] by giving only a single example of each class to train a pretrained neural network, the process is called one-shot learning. Therefore, we would like to propose a new approach for improving the performance of a deep network trained on a small training set of our microscopic blood image dataset.

1.2 Overview

We first review the influential papers proposing novel deep networks for semantic segmentation since 2015. The summary table in Appendix A, shows the datasets sizes in these influential papers. We find that there are not as many analysis on limited or small dataset cases as those for big datasets.

We conduct a breakdown experiment for demonstrating the case of training a deep network for semantic segmentation with extremely small training sets. We would like to investigate how small the size of the training set will make the recognition completely break down, and observe how the accuracy of segmentation varies when we train a deep network on different-sized training sets using some synthetic data.

We then focus on investigating whether we can train a deep semantic segmentation model on a small set of labeled images and an additional set of unlabeled images,

from our microscopic blood cell images to reach expected performance. This is because manually labeling cell marks is very time-consuming and expensive. Suppose we only create labels for a small training set. The method for utilizing also unlabeled images during the training of our deep networks is meaningful. We concentrate on red blood cells, since red blood cells are the majority of all cells in our microscopic dataset and appear in every image of our microscopic dataset. Therefore, our task of semantic segmentation is for two classes: red blood cell and others. Inspired by some works on self-training for classification problems [4, 47], we implement an EM-like self-training model based on U-Net [57] for semantic segmentation training on a small set of labeled images with a set of unlabeled images using the model’s own prediction as labels. We also propose a counting algorithm that works on output segmentation maps. In addition, we create a dot labeling tool for making dot annotations to create ground truth for validating our counting algorithm. This tool could possibly be used as an interactive labeling tool for active learning.

The contribution of our works is as follows:

1. We conduct a breakdown experiment, where we train a semantic segmentation on some very small training sets separately. We control all the settings to be the same for all the tests except for the training set size. These training sets are from the same BBBC cell dataset [32] or synthetic 2D Gaussian dataset. We use these sets of the decreasing size. We observe the model performance and show that there is no certain breakdown line for the recognition, which means there is no certain small training set size that makes the recognition break down. Remarkably overfitting occurs when the number of objects in the training set is extremely small. The testing accuracy decreased to a remarkably low value since the overfitting.
2. We propose an EM-like self-training method for doing semantic segmentation on a microscopic red blood cells image dataset. We find that results critically depend on the thresholding method related to uncertainty measure inside the pseudo-label generation function. We only find a small improvement in that regard for one of the thresholding methods. For other thresholding methods, the self-training does not improve the model performance.

3. In addition to pseudo-label generation for unlabeled data, we utilize Monte Carlo (MC) dropout [21] for assessing a confidence measure to weight the entropy of each pixel in loss function. The purpose is to give a dynamic small weight for the entropy of pixels which the model is not confident to predict. This was also aimed to increase the model performance. However, as we mentioned, training the model on most types of pseudo-label results in decreasing the accuracy.
4. We show that training the semantic segmentation model on some other pseudo-labels decrease the accuracy but result in a rapid density changing in output segmentation map. These changes help with counting the objects. Therefore, we propose a counting algorithm that works on the output segmentation map of the self-trained model. The counting algorithm ends up with a testing error of 6-9% on red blood cells. This counting method can be useful when it's hard to extract objects of interests since this method works on the predicted segmentation map and does not need any additional dot annotation. Some of these findings have been published in [44].

1.3 Thesis outline

This thesis is organized as follows: Chapter 2 provides the basic concepts of computer vision tasks and semantic segmentation is our focus. Necessary backgrounds are stated including machine learning, Neural Networks from Perceptron, MLP to Convolutional Neural Networks. Chapter 3 describes our breakdown experiments. Chapter 4 presents our approach to do semantic segmentation from a limited training set of our microscopic blood images data and provide the results and discussions. Chapter 5 summaries the thesis and discuss future work.

Chapter 2

Basic concepts

This section presents an overview of the tasks in computer vision area including localization, object detection, instance segmentation, semantic segmentation. Then we focus on talking about the concepts of semantic segmentation and deep neural networks, specifically Convolution Neural Networks as the strong model for semantic segmentation. Analysis and experiments we proposed in later chapters are conducted based on those concepts.

2.1 Computer vision task

Computer vision is a scientific field, started out in the early 1970s to mimic human intelligence of vision system for robotic system [69]. It is an interdisciplinary field for digital image processing, information engineering, machine learning, and neurobiology [68]. Computer vision today has a wide variety of real-world applications including catch match move (for movie industry), motion capture (for robots, self-driving cars), 3D-modeling, medical imaging, fingerprint recognition, etc. [69].

Recent computer vision works in conjunction with machine learning field are what we focus on. By definitions in one of the recent influential challenges of this field: ImageNet Large Scale Visual Recognition Competition (ILSRVC) challenge [60], there are four main related tasks: localization, object detection, instance segmentation, and semantic segmentation. Below is a brief summary of these tasks. Figure 2.1 shows an example of these four computer vision tasks.

(a) Localization (Single-instance localization)

Localization is the task to assign an input image one label from a fixed set of categories as well as a bounding box (x,y,w,h) containing the only one object in the image. (x,y) are the coordinates of the centre and w, h are the width, height of the bounding box. The label is the category of the only object inside the image.

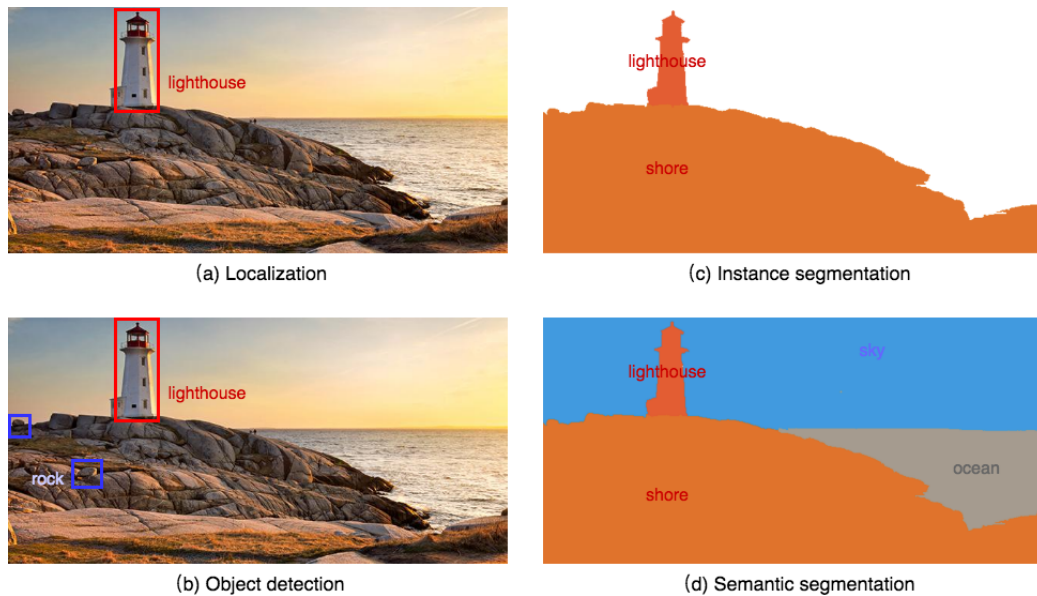


Figure 2.1: Example of computer vision tasks including object localization, object detection, instance segmentation, and semantic segmentation. (The raw image is from [74]).

(b) Object detection

Object detection is the task to find the instances of objects from a fixed set of categories in an image and output a bounding box for each instance. Suppose there are j instances, indexed by $i = 1, \dots, j$. Each of the bounding box (x_i, y_i, w_i, h_i) contains one instance of an object, labeled with the category c_i of the object.

(c) Instance segmentation

Instance segmentation is the task to detect instances of objects from a fixed set of categories in an image. For each instance of an object, all pixels of this instance are labeled with a category label. Pixels that are not recognized as instances will not be labeled.

(d) Semantic segmentation

Semantic segmentation is the task to assign each pixel in an image a label from a fixed set of categories. This task is a pixel-wise task for the entire image. Compared to instance segmentation, object level recognition is not performed. Therefore, semantic segmentation does not differentiate instances.

These computer vision tasks are easy for the human visual system to distinguish

and perform. While for machines, to completely automatically process these tasks, more works are needed. Typical ideas for solving the localization problem is to treat it as an image classification problem for deciding the class label with a regression problem for generating the bounding box. While for object detection and instance segmentation, the methods for feature extractions, the learning algorithms for recognizing and differentiating instances are most essential. Semantic segmentation problem is what we focus on in this thesis. In the next section, we discuss the details of semantic segmentation and machine learning, more specifically neural networks, as the strong solutions and models for this problem.

2.1.1 Semantic segmentation

Semantic segmentation is one of the problems in the field of scene understanding of computer vision. The task of semantic segmentation is to assign each pixel of an image an object class such as a tree, a plant, a person, a car for a dataset of nature. For a medical dataset, the class may include types of organ, tissue or cells, etc. Semantic segmentation is a pixel level task partitioning an image into parts that are semantically meaningful.

Many applications need an automatic segmentation process for 2d images, 3d images, video frames. Such applications include self-driving, face recognition for security, and robots delivery, etc. To satisfy the need for an automatic process, semantic segmentation has become one of the most frequently faced problems among computer vision problems in these applications.

Semantic segmentation could be seen as the classification for all pixels in an image. Semantic segmentation problems are being tackled using a class of models within machine learning, which is deep Convolutional Neural Networks. In this chapter, we first talk about the general supervised learning in machine learning. Then Neural Networks, most often Convolutional Neural Networks will be discussed. These are the basic concepts for semantic segmentation, which our later discussion about limited data problems is related to.

2.2 Machine learning

Machine learning is the technique to learn from training seen data to make a prediction of unseen data, where the data is a collection of vectors or matrices of real numbers, while the prediction is a class indicator vector or other vectors depending on the problem task. The learning inside is the process for enabling the model to have the prediction of unknown based on experience.

2.2.1 Supervised learning

To enable learning for a machine learning model, supervised learning means we provide the samples given to the model that include a set of example inputs and outputs. The inputs are called *features* while the outputs are called *label* or *annotation* (each pixel has a label in an image). Approaches of supervised learning are used to learn the pattern that maps the input to the output [45]. We focus on supervised learning since most methods for semantic segmentation are supervised [22]. The limited data problem we study here is mostly related to the lack of label (annotation), the high cost for creating labels, rather than the difficulty of collecting raw data.

2.2.2 Semi-supervised learning

Semi-supervised learning algorithms enable learning from a combination of labeled data and unlabeled data [11]. The model tends to have a better performance trained on the combination than just trained on labeled data. This is meaningful for reducing the cost of the whole project. Since human labeling is expensive and time consuming, utilizing unlabeled data help improve the accuracy is useful. Semi-supervised algorithms explore the unknown patterns inside unlabeled data as that information presents part of the distribution of all data in a dataset.

2.3 Neural Networks

Artificial Neural Networks are information process models for solving artificial intelligence problems. Inspired by biological neurons in the brain, a neural network is composed of artificial neurons (nodes) forming a structure to do computation through

a connectionistic approach. Used as a learning algorithm in machine learning, neural networks alter the weight and bias of each node to map the best output for the input.

Originally devised by neurophysiologist Warren McCulloch and Walter Pitts in 1943, a simple neural network is modeled using electrical circuits [48]. In 1958, resulted from the research of Frank Rosenblatt, a perceptron was built in hardware [58]. The idea of perception maintains a strong influence until today. A single-layer perceptron was found to be useful in classifying a set of inputs into one of two classes.

$$F = \sum_i w_i x_i + b_i \quad (2.1)$$

Equation 2.1 denotes a single linear layer perceptron. The output y can be made for example, by a rule: $y = 1$ if function value $F > c$, where c is a constant, otherwise $y = 0$. In 1958, the perceptron Rosenblatt proposed is the first model for learning with an error correction, which is a perceptron convergence algorithm summarized as shown below [28].

$$x(n) = [1, x_1(n), x_2(n), x_3(n), \dots, x_m(n)]^T \quad \text{input vector}$$

$$w(n) = [b, w_1(n), w_1(n), w_1(n), \dots, w_m(n)]^T$$

$$b = \text{bias}$$

$$y(n) = \text{actual response}$$

$$d(n) = \text{desired response}$$

$$\eta = \text{learning rate parameter}$$

First, the weights $W(n)$ will be initialized, where n is the index of step and m is the dimension of continuous-valued input vector $X(n)$. Further, the algorithm activates the perceptron by applying $x(n)$ and desired response $d(n)$. Actual response $y(n)$ is computed by:

$$y(n) = \sigma (w^T(n)x(n)), \text{ where } \sigma \text{ is the sigmoid function}$$

Next is the update (adaption) of the weight vector of the perceptron to obtain the followings equations. After that, increase the index of step n by one and keep doing from activation to adaption.

$$w(n+1) = w(n) + \eta(d(n) - y(n))x(n),$$

$$d(n) = +1/-1 \quad \text{if } x(n) \text{ belongs to class 1/ class2,}$$

Rosenblatt's learning algorithm above does not work for multiple layers since it adjusts the weights only for the last and the only layer in perceptron. The learning rule is devised for a single layer only, therefore lacks the method to adjust the weights before the final layer if we have multiple layers.

For more complex not linear classifiable problems, Minsky and Papert's analysis of perceptrons shows that multiple layers of perceptrons are needed [50] in 1969, which is a class of feedforward neural networks we call a multilayer perceptron (MLP) today. An MLP consists of an input layer, one or more hidden layers, and an output layer. The differences between an MLP and a single-layer perceptron are the multiple hidden layers and a non-linear activation function applied to the linear combination of node values or inputs. Non-linearity is essential for multi-layer networks since it enables the non-linear complex functional mappings between the inputs and response outputs. Equations above are commonly used activation functions.

$$\text{Binary step: } f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (2.2)$$

$$\text{Sigmoid function: } f(x) = \frac{1}{1 + e^{-x}} \quad (2.3)$$

$$\text{Tanh function: } f(x) = \frac{2}{1 + e^{-2x}} - 1 = 2 \text{ sigmoid}(2x) - 1 \quad (2.4)$$

$$\text{Rectified Linear Units (ReLU): } f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (2.5)$$

$$\text{Leaky ReLU: } f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01x & \text{if } x < 0 \end{cases} \quad (2.6)$$

$$\text{Exponential linear unit (ELU): } f(x) = \begin{cases} x & \text{if } x > 0 \\ a(e^x - 1) & \text{if } x \leq 0 \end{cases} \quad (2.7)$$

The learning algorithm for an MLP is backpropagation (BP). The idea is with a differentiable activation function for a node, derivatives can be calculated to adjust

the weights. Early BP ideas were derived from 1960s [62]. In 1974, Werbos thought to apply this principle to Neural Networks [73]. Using the chain rule, the errors are "backpropagated" from output layers to previous hidden layers and further splited and "backpropagated" from one hidden layer to another previous layers. All weights in each layer are adjusted to minimize the error. Let E denote the error. E can be a mean square error, cross entropy or other measures of the discrepancy between the predicted class and the ground truth class. If there is only one hidden layer in the MLP. The update of weight w_{ij} in Figure 2.2 is shown above.

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}} \quad \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = \sum_j \delta_j x_i$$

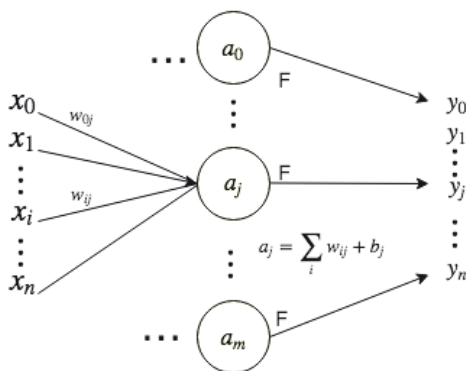


Figure 2.2: An MLP with one hidden layer. x denotes the value of input nodes, a denotes the values before activation function F of hidden nodes, and y denotes the value of output nodes.

For an MLP with multiple hidden layers, we need to consider the current node j is a neuron connected with the output layer (output node) or an inner neuron connected with another hidden layer (inner node). Suppose we have two hidden layers and the second hidden layer is connected with the output layer. The nodes we use as examples are a_j for the first hidden layer, a_k for the second hidden layer and o_k as the output of the second hidden layer. Following the chain rule, we can have the derivative:

$$\begin{aligned}
\Delta w_{ij} &= \eta \frac{\partial E}{\partial w_{ij}} & \frac{\partial E}{\partial w_{ij}} &= \left(\frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial a_k} \frac{\partial a_k}{\partial y_j} \right) \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \\
& & &= \left(\sum_k \delta_k y_j w_{jk} \right) \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \\
& & &= \left(\sum_k \delta_k y_j w_{jk} \right) F'(a_j) x_i
\end{aligned}$$

This two-layer MLP demonstrates two cases of nodes: inner node and output node. To extend the case to arbitrary multiple hidden layers, for convenient expression, let δ be: $\delta_k = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial a_k}$, an example for output neuron, and $\frac{\partial E}{\partial w_{ij}} = \delta_j x_i$: an example for inner neuron. Suppose t is the ground truth.

$$\delta = \begin{cases} \text{error}(y_j, t_j) f'(a_j) & \text{when } j \text{ is output neuron} \\ \left(\sum_j \delta_j y_j w_{ij} \right) f'(a_j) & \text{when } j \text{ is hidden neuron} \end{cases}$$

2.3.1 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a class of neural networks, mostly applied to visual content such as images, video frames, signals, etc. CNN is well known for success at vision-oriented classification tasks [40, 38]. CNN is composed of convolution layers for generating feature maps, downsampling layers for feature map size reduction, and fully connected layers as same as an MLP. Compared to MLPs, CNNs are built specially for computer vision problems. Instead of extracting the features in an image by all the neurons, convolution function is used in neurons to reduce the complexity of neural networks. Also, convolution layers in CNNs enable more efficient learning compared to an MLP, about learning features from objects in different positions.

For CNNs, the feature map is generated by passing a convolution matrix (kernel) to the input matrix map (input image), multiplying each entry in the kernel size and summing. The kernel can be seen as weights for adding neighbor entries for an entry in an input matrix map (input image) and the weights are shared across all patches of the map. The weights updating is invariant to the position. Therefore CNN has translation invariance characteristics, which means the learning will be the same regardless of object positions. Since convolution layers are consecutive and

latter layers made compositions of lower level visual features from previous layers, the output of the last convolution layer will be deep hierarchical features that are capturing the whole semantic information.

The output layers (fully connected layers) enable classification by connecting all neurons to the last output layer, which contains the same number of neurons as the category of classes. Classification is performed by taking the corresponding class with the highest normalized, for example, Softmax [7] probability.

2.3.2 Neural Networks for semantic segmentation

The general architecture of neural networks for semantic segmentation consists of encoder layers followed by some decoder layers. Encoding is the process as same as hierarchical feature extraction and feature map size reduction in a CNN. Instead of a classification decision as an end result where the fully connected layers in a CNN do, the decoding is to enlarge the feature map size and project the features extracted onto the pixel space, therefore recover the morphological details of objects.

For encoder layers, any architecture of convolution and pooling for feature extraction in a CNN pretrained for classification can be used, such as AlexNet [38], VGG [67] or ResNet [30]. Decoder layers will be the mirroring layers of encoders replacing poolings with deconvolution (or called transposed convolutions).

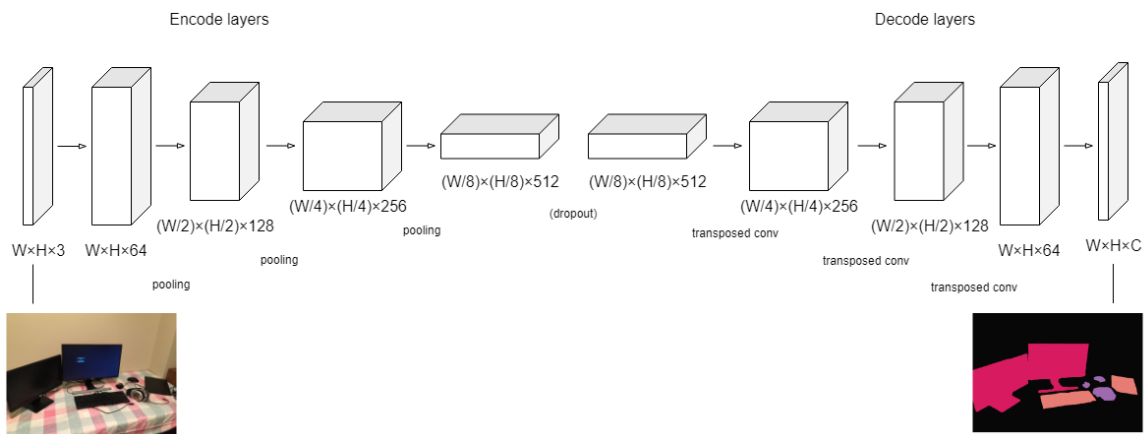


Figure 2.3: Example of a basic structure of a encoder-decoder network for semantic segmentation. (W , H denote the weight and height of the input. C denotes the number of classes).

The early papers that popularize the encoder-decoder structure are [64, 43]. They

observe that the fully connected layers of a CNN can be viewed as a patches-sharing efficient computation equivalent to convolving the whole image input. Therefore, the compressed hierarchized features maps in fully connected layers can be used to gain (or called recover) feature heatmaps same size as input. Then classification decision can be made on the feature heatmap for each pixel. In an encoder-decoder neural network, the layers before fully connected layers are encoder layers, while the layers after fully connected layers are decoder layers.

The method of recovering the feature heatmap is to skip mapping fully-connected layers to a class vector. Instead, add some structures to interpolate the small size feature maps so that the feature maps will be up-sampled till the same size as the input image. The structures are usually transposed convolution (deconvolution) layers with strides. The stride size is corresponding to the stride size of poolings in encoder layers. In [43], their implementation of an encoder-decoder network is called Fully Convolutional Network.

Later encoder-decoder networks introduce adding some shortcut connections between encoder layers and decoder layers. Some information is copied from encoder layers and attached to decoder layers. The purpose is to get better recovered details during upsampling the feature map.

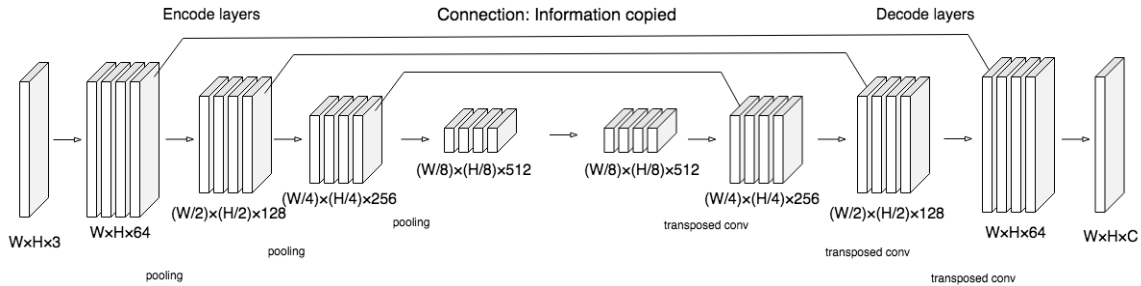


Figure 2.4: Encoder-decoder network with shortcut connections

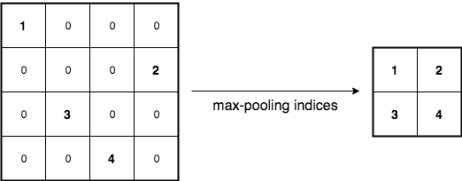


Figure 2.5: Maxpooling indices are stored during the feature extracting

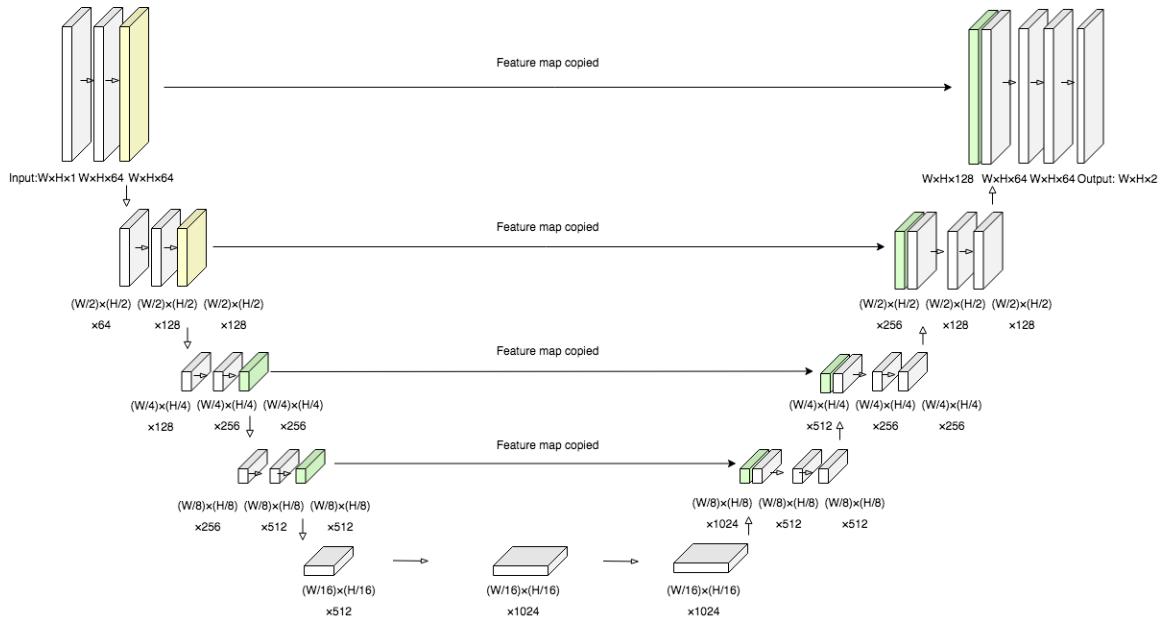


Figure 2.6: The architecture of the U-Net. Entire feature maps are copied and concatenated from encoder to decoder layers

In SegNet[3], indices from maxpooling are copied. They use these indices to up-sample the feature maps. Another encoder-decoder network famous for biomedical images is U-Net[57]. Instead of copying maxpooling indices, U-Net copy and concatenate the entire feature maps from encoder layers to up-sampled feature maps in corresponding decoder layers, and perform transpose convolution in the decoder layers.

2.4 Evaluation measures for semantic segmentation

Evaluation measures are important when it comes to our study goal: the relation between the performance of the semantic segmentation and the size of data or label. Existing measures of semantic segmentation includes region based measure, contour-based measures.

2.4.1 Region based measure

Most evaluation of measures for semantic segmentation focus on pixel level classification accuracies. These measures are region based since all pixels of an input image are evaluated and pixels forming regions correspond to objects such as human, plant,

vehicle, animal, cell, building, etc., and scenes such as street, sky, lawn, cytoplasm, etc.

In the paper of Fully Convolutional Networks [43], four evaluation metrics used in the paper become widely used in papers of semantic segmentation. These metrics are summarized in Equation 2.8 - 2.11 below. Mean Intersection over Union (Mean IoU, Jaccard index) and Dice coefficient (F1 score) are two most widely used metrics. The following notations are used: n_{ij} : the number of pixels of class i predicted to belong to class j , where there are n_{cl} different classes. t_i : the total number of pixels of class i .

$$\text{Pixel accuracy (Overall Pixel)} = \sum_i n_{ii} / \sum_i t_i \quad (2.8)$$

$$\text{Mean accuracy (Per Class)} = (1/n_{cl}) \sum_i n_{ii}/t_i \quad (2.9)$$

$$\text{Mean IoU (Jaccard index)} = (1/n_{cl}) \sum_i n_{ii} / (t_i + \sum_j n_{ji} - n_{ii}) \quad (2.10)$$

$$\text{Frequency-weighted IoU} = (\sum_k t_k)^{-1} \sum_i t_i n_{ii} / (t_i + \sum_j n_{ji} - n_{ii}) \quad (2.11)$$

$$\text{Mean Dice coefficient (F1 score)} = (1/n_{cl}) (2 \times \sum_i n_{ii}) / (t_i + \sum_j n_{ji}) \quad (2.12)$$

Semantic segmentation classifying each pixel of an image can be considered as a process of three steps [23]: object detection, shape recognition and classification. Many studies and researches are specially and explicitly focused on improving the accuracy of the last two steps by highlighting the improvement of the IoU, or Dice coefficient [22]. In term of the statistical measure, IoU is $\frac{tp}{tp + fp + fn}$ while dice

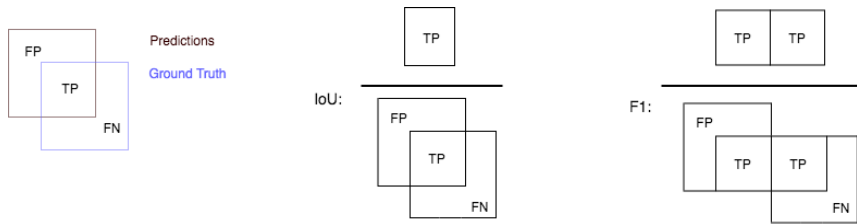


Figure 2.7: Simple bounding box example of IoU and F1 measures: ratios of statistical measures.

coefficient is $\frac{2tp}{2tp + fp + fn}$. Their function is equivalent in terms of measuring score averaging over all classified pixels.

The difference is that the IoU metric tends to penalize more to single incorrectly classified instances while the Dice coefficient tends to penalize less as shown in the equation below.

$$\text{Dice coefficient (F1)} = \frac{2IoU}{IoU + 1} \quad \text{therefore} \quad \frac{F1}{2} \leq IoU \leq F1$$

Region based measures are commonly used for benchmarks and challenges of semantic segmentation. Microsoft COCOs standard metric [41], simply denoted as $mAP@[0.5, 0.95]$, is calculated by averaging mean Average Precision (mAP) over IoU thresholds the segmentation probability output map from 0.5 to 0.95 with step 0.05. PASCAL VOCs metric is $mAP@0.5$ [19].

2.4.2 Contour based measure

For some applications of segmentation, the contour quality significantly contributes to the perceived segmentation quality [36]. Popular contour-based measures such as [46, 20] are based on doing the closest match between boundary points in the prediction and ground truth segmentation maps. In [46], they do an F1-measure using set a distance error tolerance θ for checking if a point of the boundary has a match. In [16], they further extend this to an F1-measures contour based score in semantic segmentation as (2.13) shows. For a class c , calculate the precision (P^c) and recall (R^c) by comparing contour map for the binarized predicted segmentation map with the ground truth contour map.

$$F_1^c = \frac{2 \cdot P^c \cdot R^c}{P^c + R^c} \quad (2.13)$$

2.4.3 Benchmark dataset for semantic segmentation

During the development of deep learning model for semantic segmentation, when novel models or extensions are proposed, some datasets are used as standard benchmarks. We will list some most influential benchmark datasets for semantic segmentation below.

- (a) The PASCAL Visual Object Classes(VOC) 2012 [19]: This dataset contains images of 20 classes: person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tv/monitor. For semantic segmentation challenge, the training set and validation set contain 1464 and 1449 images respectively. The annotations are full pixel-level.
- (b) Microsoft Common Objects in Context (COCO [41]): This dataset is large-scale and contains very diverse objects. There are over 80 classes including person, transportation: car, bike, bus, truck, boat, train, plane, etc., animals: bird sheep dog, etc., fruits, electronics, sports equipment etc. There are 80k images for training and 40k images for validation and the dataset keeps growing. COCO is famous for its large scale and wide range of object classes.
- (c) Cityscapes [14]: This dataset contains street scene images in 8 classes: humans, vehicles, flat surfaces, constructions, objects, nature, sky, and void. There are 5k images with fine annotations and 20k coarse annotated images. The layouts and locations of objects on the street are diverse. This dataset is influential for projects related to driving.
- (d) PASCAL Content [52]: This dataset provides an extension of PASCAL VOC 2010, adding images to 10k in 540 classes (60 frequent classes + 480 rare classes). Much more classes are created for the background in original PASCAL VOC dataset. This dataset focuses on setting more complex scenes that more difficult to generalize.

- (e) ventral nerve cord(VNC) medical dataset [9]: The dataset contains 30 images (512x512 pixels) from a serial section Transmission Electron Microscopy (ssTEM) data set of the Drosophila first instar larva ventral nerve cord (VNC). Each image comes with a corresponding fully annotated ground truth segmentation map for cells and membranes.
- (f) NYU Depth Dataset V2 (NYUDv2) [65]: This dataset contains video sequences from a variety of indoor scenes across 26 scenes, where there are 35,064 distinct objects of 894 different classes (table, chair, bed, canibet etc.). The dataset consists of 1449 RGBD images, gathered from a wide range of commercial and residential buildings in three different US cities.
- (g) SIFT Flow [42]: This dataset contains outdoor scenes including streets, mountains, fields, beaches, etc. There are 2688 (256×256) images of 33 classes. The dataset has a number of issues, including unannotated images and missing classes from the test set.
- (h) CamVid road scenes [8]: This dataset contains road scenes captured by cameras mounted on a car. The video sequences are sampled adding up to 701 frames (images). Those images are manually annotated with 32 classes including void, building, wall, tree, vegetation, etc.
- (i) Person-Part dataset [13]: The dataset provides images and pixel-level labels for six person parts including Head, Torso, Upper/Lower Arms and Upper/Lower Legs. The rest of each image is considered the background. There are training 1717 images and 1818 test images.
- (j) ADE20K [77]: The dataset focuses on scene parsing or objects and stuff recognizing in an image. The images are both indoor and outdoor. There are 20,210 images for the training set, 2,000 images in the validation set, and 3,000 images in the testing set. All the images are exhaustively annotated with objects. Some objects are even annotated with their parts. For example, the chair object class consists of chair-back, chair-seat, chair-leg, etc.

- (k) Semantic Boundaries Dataset (SBD) [27]: This dataset is an extension of PASCAL VOC, providing category-level and instance-level annotation without boundaries (Original annotations contain boundaries in PASCAL VOC) for 11k images from PASCAL VOC 2011. This dataset is divided into 8498 images for training and 2957 images for validation.
- (l) Stanford Background dataset [25]: The dataset contains outdoor scene images from multiple public datasets including LabelMe, MSRC, PASCAL VOC, and Geometric Context. There are 715 (320×240) images and the number of classes in annotations depends on the algorithm provided in their paper to generate the labels differentiates the objects, regions and geometry of a scene.

Most standard benchmarks need to be diverse and representative enough to judge a model’s general performance. When a novel technique or extension is proposed, to make fair comparison between sophisticated neural networks possible, existing standard representative datasets are necessary. Standard large-scale datasets are convincing and representative enough for some of real scene in applications. Those benchmark datasets are growing with more data and evolve over time so they end up with a large number of samples till today. Table 2.1 summarizes the data quantity growing of Pascal VOC and MS COCO datasets.

Pascal VOC data sets			MS COCO datasets		
Year	class	size (the number of images - objects (+ segments))	Year	class	size
2005	4	1578 - 2209	2014	91	83K(train) + 41K(val/test)
2006	10	2618 - 4754	2015	91	81K(test)
2007	20	9,963 - 24,640	2017	91	118K(train) +5K(val) +41K(test)
2008	20	4,340 - 10,363			
2009	20	7,054 - 17,218 + (3,211)			
2010	20	10,103 - 23,374 + (4,203)			
2011	20	11,530 - 27,450 + (5,034)			
2012	20	11,530 - 27,450 + (6,929)			

Table 2.1: The growing dataset size of Pascal VOC and MS COCO benchmarks

2.5 Limited data problem

In order to properly understand how semantic segmentation is influenced by limited data, we first outline what we mean with a limited data problem. A dataset can be limited in a variety of ways.

Limited dataset size

The size of a dataset can be limited, which means there is a very small amount of raw data. The data is insufficient for training a model to get a performance goal. Although many standard benchmark datasets have been grown to be large scale, real-world problems are different. For a specific research or application, it's possible to have small size dataset due to the sample resources limit, technical limit or time cost requirement for collecting data. For example, in some competitions of the robotic solution, collection and labeling need to be performed in a short available time. The robot will work in several scenes and need to make a quick collection of photos used as data for future behavior [49]. In the medical domain, many datasets contain only a small number of training samples, such as Drosophila first instar larva ventral nerve cord (VNC) dataset in the U-Net paper contains only 30 images (512x512 pixels). In Broad Bioimage Benchmark Collection [32], there are over 40 medical image datasets including datasets of human red blood cells, Human Hepatocyte, Mouse trophoblast stem cells, etc. Most of these datasets contain less than 200 sample images.

Limited label

Labels for semantic segmentation can be limited. Label (annotation) for an image is a separate map with the same resolution as the image data. Intuitively, it means every pixel has its own class number as label. The class label refers to its enclosing object or scene. The labels can be limited in quality or quantity.

Label quantity Lack of labels is the most common problem after collecting the data, because the financial cost and time cost for creating labels are higher than collecting raw images. There are many labeling tools for helping create annotations for images. [17]. Nevertheless, creating labels is currently not completely automated and is done by manual work mainly. Creating labels manually is expensive. In the medical domain, label creating is even more expensive compared to natural datasets. It needs to be precise and often require domain experts to do it professionally and attentively, such as human cancer cell image or other image data for medical diagnoses.

Label quality In addition to the quantity of labels, the quality of labels can be limited. Labels can be coarse, incomplete or mis-labeled. The quality of label represents how accurately the annotation matches the shape, structure and boundaries of the object. This is related to the features, patterns extracting and decision making for a neural network. Since it is time-saving and cheap to produce a low quality approximate label, imperfect labels are common in public natural datasets such as the MS COCO and the Cityscapes, etc. In the Cityscapes dataset, for example, there are fine annotations for around 5000 images and about 20000 coarsely annotated images. For custom datasets where different people collect image data and creating labels for their own, it is very possible to have imperfect labels since object boundary is difficult to accurately annotate.

The term of coarse label here refers to the fact that the label locates objects well but not accurately represent the boundaries. A typical case is that of a polygon enclosing an object. An incomplete label means the annotation is incomplete so that some instances of objects missed being annotated in the annotation image. Except for coarse labels and incomplete labels, mis-labeling can happen in every dataset. Proposing new models to learn from imperfect labels is worth doing. Some deep learning models are shown to be robust when trained on some datasets with noise labels and errors [56].

2.5.1 Labeling data

Utilizing deep learning requires the labels(annotation) for data as most deep networks are supervised or semi-supervised. The cost of doing labeling is related to the targeting computer vision task. For some tasks, we need accurate pixel-wise annotation of objects. Others may just need objects to be labeled by landmarks or bounding boxes.

Table 2.2 provides a summary of required label type and cost estimation for tasks in the field of Computer Vision. We make an estimate of the cost of labels, label quality demanding, graded starting from one to several stars, depending on how much difficulty and time consuming for putting works on it. We also mention some online labeling applications for these computer vision tasks.

Computer Vision Task			Labeling tools	Labeling cost	Label quality demanding
		Output			
Image level	Localization	Single object, bounding box	OpenLabeling [10], Daturks [18]	*	*
	Object detection	Multiple objects, bounding boxes		**	*
Pixel level	Instance segmentation	Multiple objects, pixel-wise annotation	Labelbox [39], LabelMe [61]	***	**
	Semantic segmentation	Multiple objects and scenes, pixel-wise annotation		***	**

Table 2.2: Summary of the required label types and cost estimation for computer vision tasks

2.5.2 Using outside data resources

In addition to the type of targeting computer vision task, the challenge we face in real-world applications is that data has different representation in different domains. Domain refers to the specialized background field of data. There is a strong resemblance between the information represented for the data in the same domain.

Application	Domain	Related datasets
Self-driving, navigation	Driving	Cityscapes, CamVid, SYNTHIA, KITTI
Household object capture	Indoor	NYUDv2, SUNRGBD, rgb-d object dataset
Natural object capture	Outdoor	SIFT Flow
Face recognition	Portrait	Adobes Portrait Segmentation, CASIA, MS-Celeb-1M
Investigate	Material	Materials in Context Database (MINC)
Diagnosis	Medical	

Table 2.3: Domains of applications and the related datasets

Limited data problem tends to be more challenging in uncommon domains than the domains that are fully supplied with domain expertise. When the source data is limited, the problem can be tackled by utilizing outside resources, if the domain of dataset that needs to be worked on is common and has related public sources. Pretraining or domain adaption are the techniques that can be applied to enable utilizing the outside related datasets. Table 2.3 shows some specialized purposes applications and their domains.

In addition to the dataset itself, domain knowledge helps make sure whether data

is enough to reasonably select meaningful features from source data and find the relationships between source data and target data. Utilizing outside data and domain expertise should be considered and may substantially improve the predictive performance. That is essential before confirming that more data must be collected and more labels must be created.

Chapter 3

Breakdown Experiment

3.1 Motivation

In general, semantic segmentation with deep learning has enormous success when the dataset is large-scale, and most of the popular benchmark datasets as described in Chapter 2.4.3 are large-scale. There is no a common standard for claiming a dataset is large-scale or not large-scale. Here we describe large-scale by meaning the number of images in a dataset is more than one thousand, and usually the unit of measurement for the data in that dataset is thousand(k). For example, the training set of PASCAL VOC 2012 for segmentation contains 1.4k images, and the training set of Cityscapes dataset contains 3k images.

Influential deep learning semantic segmentation methods surpassing the former methods in recent years, are evaluated using the popular benchmarks. We made a summary table of the datasets and the number of data images used in the papers of the most influential deep learning methods for semantic segmentation. Most of the datasets used in the papers are large-scale. The table is located at Appendix A.1. These papers lack the results for how the performance of their models will be, if the model is trained on data that is less than they used for their experiments. This is the reason for us to analyze the case of limited data for deep network methods.

For many machine learning algorithms, estimation of how much data is required to achieve an expected performance goal has been studied. Support Vector Machine [15] as an example, is a model classifying data by constructing a set of hyperplanes, mapping data points into a high dimension space. In that space, the SVM learns from training data for making the hyperplane have the largest distance (margin) between all training data points of all classes. The amount of training data needed for an SVM classifier depends on the complexity of the classifier. Vapnik – Chervonenkis (VC) dimension, defined as the size of the largest set of points that the classifier can shatter, is introduced to measure the the complexity of the classifier. For example,

suppose we know the number of the features or dimensions of the data. To map the data into an even higher dimension, we set some hyperparameters of the SVM to make the generalization abilities higher. How much data we need to learn for an SVM is about how many mistakes for a number of samples will make when SVM learning converges. The bound is about how many mistakes will be is less than a bound that is related the to the number of samples [5].

$$|\hat{\epsilon}(h) - \epsilon(h)| \leq O\left(\sqrt{\frac{d}{m} \log \frac{m}{d} - \frac{1}{m} \log \delta}\right) \quad (3.1)$$

Equation 3.1 is a theorem of agnostic learning of VC bounding. Suppose m is the number of points, d is the VC dimension, h is one model (classifier) from the model functions family H . And ϵ is the error for testing data, $\hat{\epsilon}$ is the error for training data. Their difference is the final mistakes when the SVM converges. Then with probability at least $1-\delta$, for all h belongs to H , the mistake bound is less than the right-hand side of the equation, where O means space complexity. To guarantee any hypothesis that perfectly fits the training data is with probably $1 - \delta$ have error tolerance rate ϵ on testing data from the same distribution. The number of training sample m should be as equation 3.2 shows [51].

$$m \geq \frac{1}{\epsilon} (4 \log_2(2/\delta) + 8VC(H) \log_2(13/\epsilon)) \quad (3.2)$$

Compared to other machine learning models, a deep learning model has a huge number of parameters. For example, The total number of parameters in AlexNet is 62 million, and the 16 layers VGG convolution network has over 138 million parameters. Suggesting the smallest training data size for training a deep network to achieve a performance goal is a complex problem. Therefore our experiments aim only at exploring the cases for some very small training sets, since the scenarios exist that we have to apply a novel deep learning technique to the current dataset of a small size.

Our experiment consists of many tests, where we train a deep network with training sets of different sizes separately. We summarize the performance to explore how the model performance decreases as the reduction of training set size. We would like to investigate how testing accuracy is related to training set size, and how small the training set will make the recognition of the objects broken.

3.2 Related work

Some related works have been done for predicting how accuracy varies with training data size, such as [33]. They demonstrate the accuracy versus training data size for MNIST classification. The model is a weighted least squares regression. The classifiers are a shallow CNN and a deep CNN. They have three proposals for how error depends on training data size n (b, c are two variables): $\text{Error} = bn^{-c}$, $\text{Error} = a + bn^{-1/2}$, $\text{Error} = a + bn^{-c}$. These three proposals are used to extrapolates the testing results of the classifier. They plot an error vs training size graph where each dot represents an accuracy test on the same testing set run on a different-sized subset of training data. Error axis has a linear scale, training size axis has a log scale. They show the linear regression line of three proposals. Their purpose is about predicting training data requirements while our research focuses on how the generalization will be for a deep network trained on the extremely small dataset size.

3.3 Methods

We set up a semantic segmentation model and keep using the same configurations for running each separate test. The architecture we use is the U-Net. For each test, we train the model for the same 300 iterations and validate the performance on the same testing data. The only control variable is that, for each test, we have a different training set size: the number of training images, the resolution and the number of objects. We have a group of training sets, where the one of largest size is a training data size that enables the model to achieve a sufficient performance. Here we mean sufficient by a performance the deep network model can recognize most of the objects while could be having some errors in the shape of the objects or type of objects. With regards to accuracy, we choose a mean IoU of around 50% to be our sufficient performance.

We reduce the size of that first largest training set to make the rest training sets, until the training set becomes extremely small with only very few objects in it. To summarize the model performance in each test, we generate the curves for testing accuracy against training set size. We also demonstrate some model predicted map of testing images corresponding to the size of the training set, visually showing some

examples of errors.

Datasets

The datasets we use for our first experiment is a dataset of cell images from Broad Bioimage Benchmark Collection (BBBC) [32]. The images represent some clustered nuclei, with annotation of two classes: 0.background, 1.nuclei. There are 100 images in total, where each image with resolution (950×950) contains exactly 300 cells (nuclei).

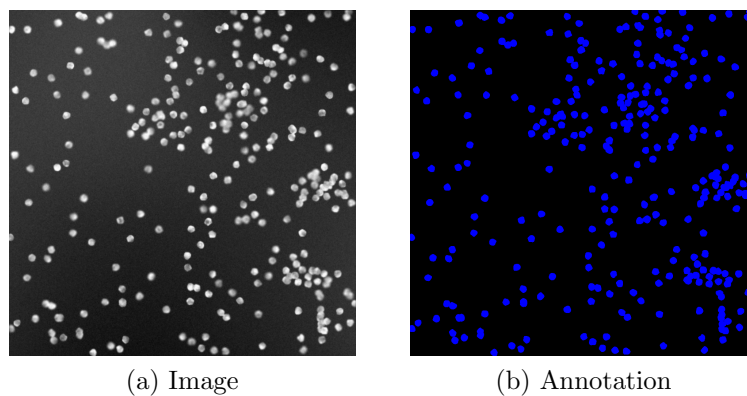


Figure 3.1: Example image and ground truth annotation of BBBC cells data

The other dataset we use is a synthetic dataset where we generate a certain number of 2D-Gaussian objects at random positions in an image with Gaussian noisy as background. The standard deviation of those 2D-Gaussian objects are in range 1 to 5 pixels. We save the coordinate of the centre of each Gaussian object after they are generated.

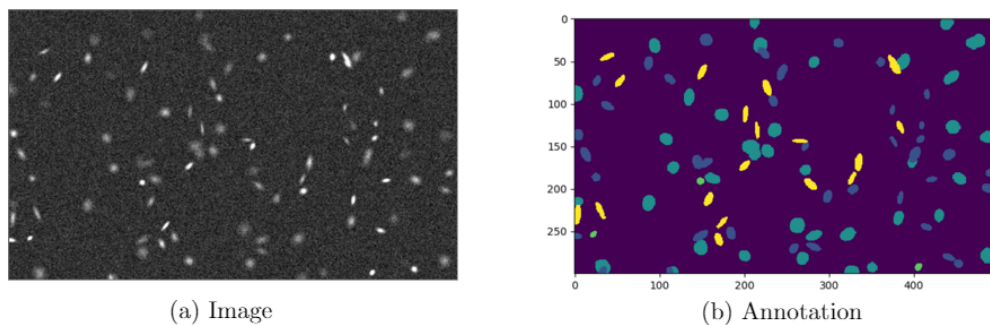


Figure 3.2: Example image and ground truth annotation of synthetic 2D Gaussians dataset (100 Gaussian objects)

class background	None
class 0. Normal	all Gaussian objects not class 1.2.3.
class 1. Big	$x_y_stddev > c_big$
class 2. Small	$x_y_stddev < c_small$
class 3. Long	if not big, not small, and $ x_stddev - y_stddev > c_long$

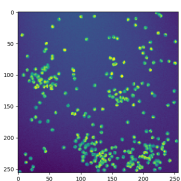
Table 3.1: Five classes of the synthetic 2D-Gaussian dataset, where c_big , c_small , c_long are constants for thresholding when generating the Gaussian objects

We divide the objects into 5 classes depending on the size and shape and generate their corresponding annotations as shown in Table 3.1 and Figure 3.2. If the standard deviation (stddev) of both horizontal and vertical are large than a constant c_big , the object is set to class 1: Big. In contrast, if both are smaller than a constant c_small , the object is set to class 2: Small. Objects that are not big or small and the difference between its horizontal and vertical stddev is larger than a constant c_long , are set to class 3: Long. All others are set to class 0: Normal.

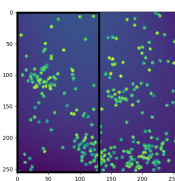
Experiment

The only variables we control here is the size of the training set: the number of training images, the size (resolution) of them and the number of instances in those images. Table 3.2 demonstrates examples of the training data sizes in a decreasing order.

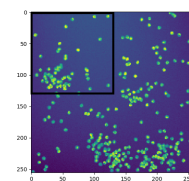
Dataset 1: BBBC synthetic cell image



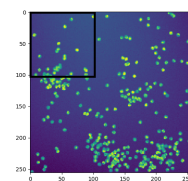
1-256x256-300



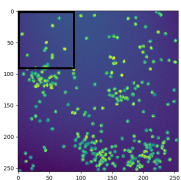
1/2-128x256-130±



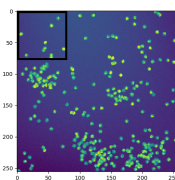
1/4-128x128-60±



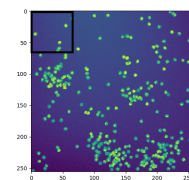
1/6-102*102-27



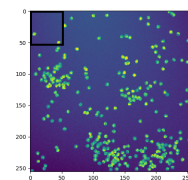
1/8-90x90-12



1/12-77x77-10



1/16-64x64-5



1/25-50*50-2

 Dataset 2: Synthetic 2D Gaussian dataset

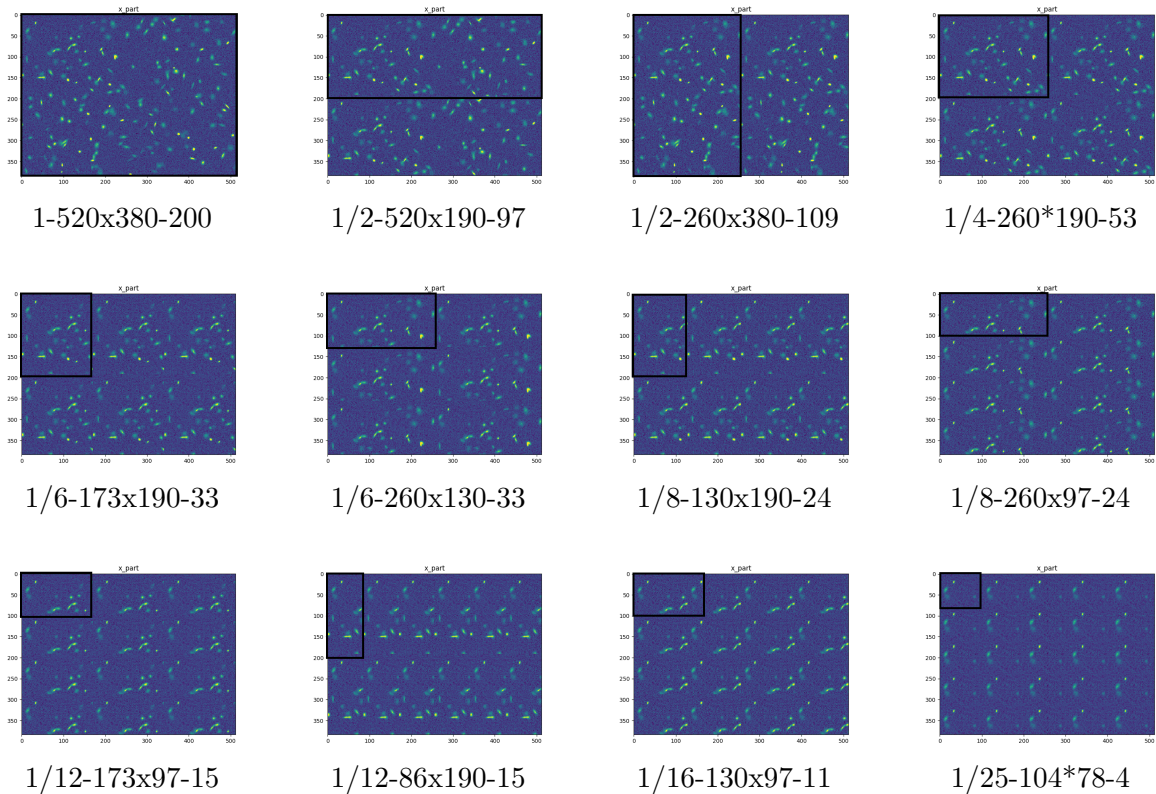


Table 3.2: Examples of the training data in descending order of training size in configurations: [number of images - resolution - number of instances]

As shown in Table 3.2 shows, We augment the cropped part of an image (also annotation) to its original resolution by repeating this sub-image filling the output with its repeated copies. For example, if we use a quarter (260×190) of a 520×380 image, there will be four same sub-images filling a 520×380 image. These augmented images are used for training.

Table 3.3 presents the class balance of all the training sample configurations. The first left column indicates the cropped parts corresponding to configurations as shown in Table 3.2.

Cropped part fraction-(height,width)	1 image	3 images	8 images
1-(1.0, 1.0)	[130 21 26 23]	[404 60 68 68]	[1055 163 192 190]
1/2-(0.5, 1.0)	[60 10 11 16]	[193 26 35 36]	[526 80 100 94]
1/2-(1.0, 0.5)	[73 11 16 9]	[209 31 38 34]	[518 81 104 85]
1/4-(0.5, 0.5)	[34 4 8 7]	[94 10 21 17]	[245 39 62 42]
1/6-(0.5, 0.3333)	[21 2 7 3]	[60 5 15 8]	[163 26 39 28]
1/6-(0.3333, 0.5)	[20 4 5 4]	[60 8 14 11]	[148 31 42 23]
1/8-(0.5, 0.25)	[16 2 3 3]	[43 5 10 7]	[117 24 27 23]
1/8-(0.25, 0.5)	[14 3 4 3]	[47 6 12 8]	[117 22 31 19]
1/12-(0.25, 0.3333)	[9 1 4 1]	[31 2 9 2]	[77 14 21 12]
1/12-(0.5, 0.1667)	[9 2 2 2]	[24 3 5 6]	[70 18 18 18]
1/16-(0.25, 0.25)	[7 1 2 1]	[23 2 6 2]	[52 12 15 10]
1/25-(0.2, 0.2)	[1 1 2 0]	[12 2 5 0]	[31 8 13 6]

Table 3.3: The number of Gaussian objects and their class distribution in training samples in format[class 0: Normal class 1: Big class 2: Small class 3: Long]

3.4 Results

In this section we present the results of our break down experiment and our observations from the predictions of the testing images.

Dataset 1: BBBC cells images dataset

The training and testing accuracy of mean IoUs are shown in Figure 3.3. Mean IoU axis has a linear scale, while number of training objects axis has a log scale. Overfitting occurs since there exists a generalization gap between the training accuracy curve and the testing accuracy curve, especially when the number of objects is very small.

We compared our testing results with the proposals in [33] for how error depends on training data size n : $\text{Error} = a + bn^{-1/2}$, $\text{Error} = a + bn^{-c}$. The Pearson coefficient of correlation (r) between the testing curve and the square-root relationship accuracy $= 1 - (a + bn^{-1/2})$ is 0.955. The Pearson coefficient of correlation (r) between the

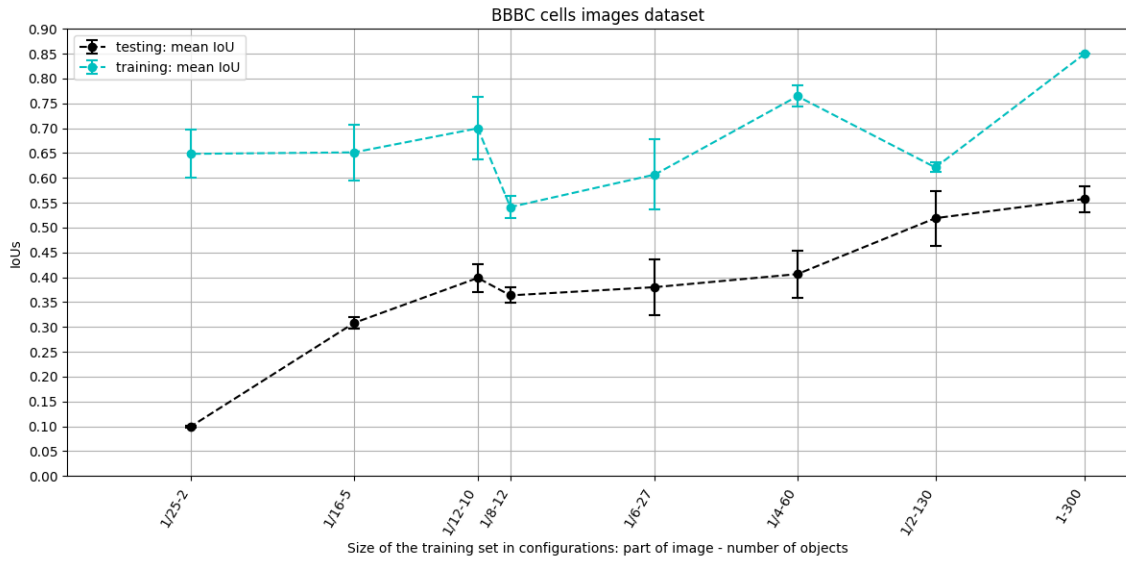


Figure 3.3: The results of training and testing accuracy of mean IoUs given the training sample size and how many objects in training images, tested on 10 testing images (950×950), averaged over 6 runs. Variances are shown as errorbars

testing curve and the extended power law relationship $accuracy = 1 - (a + bn^{-2})$ is 0.875. The testing curve and the proposed relationship curves has the strong linear association. Then we did a least square regression to find the best a and b fitting our testing curve.

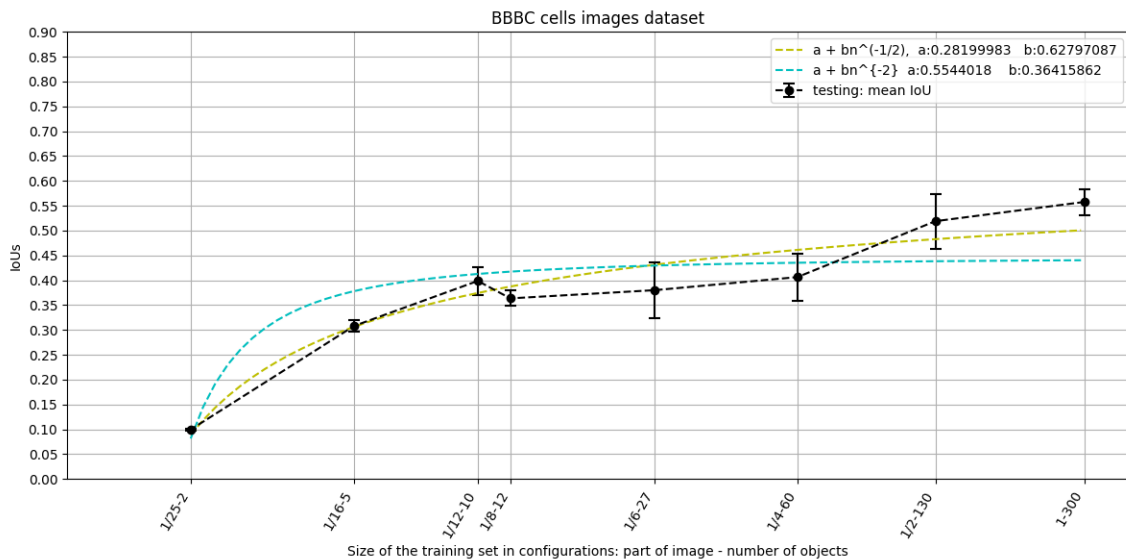


Figure 3.4: Testing accuracies with the regression line of proposal relationship: $accuracy = 1 - (bn^{-1/2})$, and $accuracy = 1 - (a + bn^{-2})$

In Figure 3.4, the yellow dash line shows the least squares regression of the proposal accuracy = $1 - (a + bn^{-1/2})$. The regression coefficients result is a=0.282, b=0.628. The cyan dash line shows the least squares regression of the proposal accuracy = $1 - (a + bn^{-2})$. The regression coefficients result is a=0.554, b=0.364. This simple test for comparing our results with those proposals shows that the power-law or the square-root relationship cannot perfectly fit the testing curves and cannot be a bound for the relationship between accuracy and training set sizes in our semantic segmentation experiment.

Here we focus on finding how small the training set will make the recognition break down. By checking the outputs, we find that training the networks on our smallest training set: 1/25 of one full image containing only 2 cells, results in predicting the whole testing image to all background class for most of the testing samples. However, as shown in Table 3.4 (case 1), for a few samples in some runs, the model predicts the testing images with some correct segmented cells. Therefore, the recognition of the model does not completely break down. The model overfits to this extremely small size unbalance training set, where the only 2 cells in the training image take up very small proportions of the pixels, while all other pixels are background class.

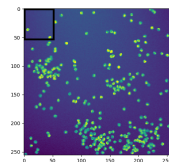
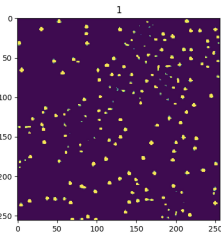
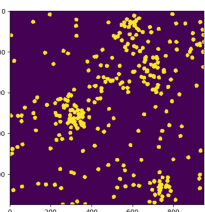
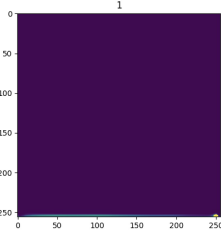
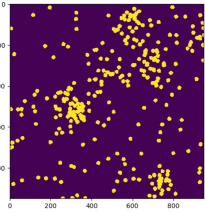
Training set size	Prediction	Ground Truth
		
1/25-50x50-2	Case 1, appears in 2 of 6 runs, only for a few testing images.	
		
	Case 2, appears in all 6 runs, for most of the testing images.	

Table 3.4: Example of the inference results of a testing image of the model trained on 1/25 of a full image (resolution 50x50) containing only 2 cells.

Dataset 2: Synthetic 2D-Gaussian dataset

In each test, we test on 10 testing images (520×380) for the synthetic 2D-Gaussian dataset. Figure 3.5 shows a curve of IoUs against the sizes of sampled training data. These three curves correspond to three groups of different-sized subsets of training sets. These three training sets contain 1 image, 3 images, 8 images relatively. The horizontal axis label represents how large the parts are. The parts mean that we use a cropped fraction of each of the training image in the set. For example, $1/4$ of 3 images with resolution (520×380) means we use the upper-left corners (260×190) of each of the 3 images for training.

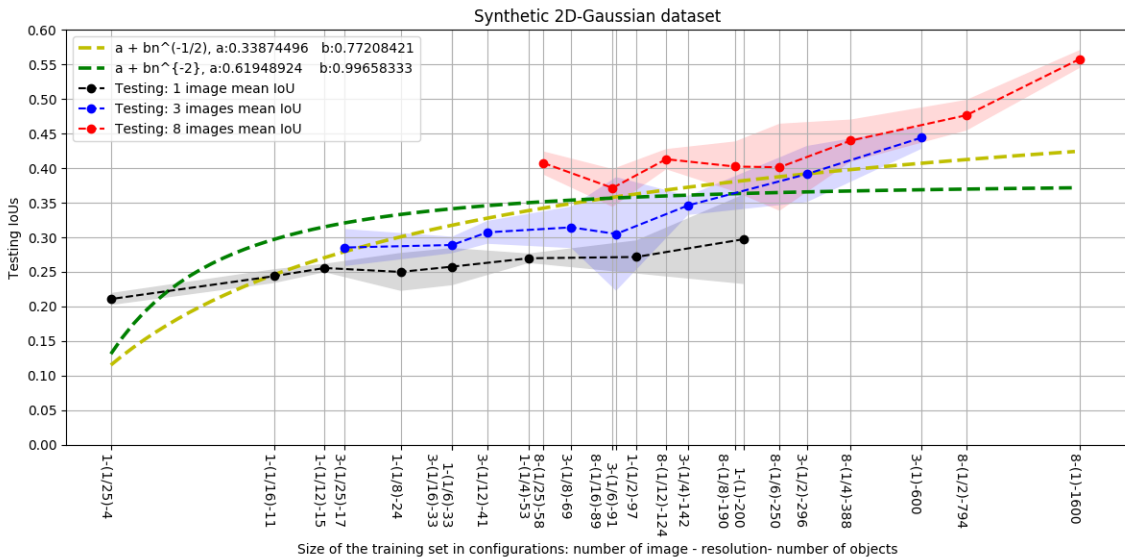


Figure 3.5: Testing results for mean IoUs given the training sample size in configurations: [number of images - [fraction part size] - number of instances], averaged over 6 runs. Variances are shown as shaded

We compared our testing results with the proposals: Error = $a + bn^{-1/2}$, Error = $a + bn^{-c}$. The square-root relationship accuracy = $1 - (a + bn^{-1/2})$, has the Pearson coefficient of correlation (r) of 0.761. The extended power law relationship accuracy = $1 - (a + bn^{-2})$ has the Pearson coefficient of correlation (r) of 0.555. The testing curves and the proposed relationship curves have less linear association than those of dataset 1.

In Figure 3.5, the yellow dash line shows the least squares regression of the proposal accuracy = $1 - (a + bn^{-1/2})$, $a = 0.339$, $b = 0.772$. the green dash line shows the least

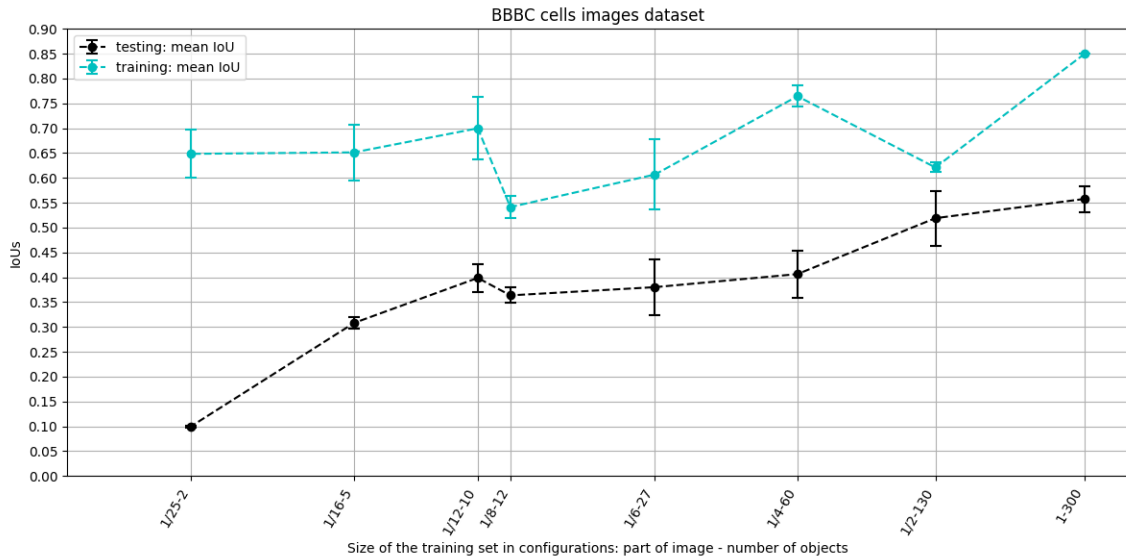


Figure 3.6: Training and testing results for mean IoUs given the training sample size and how many objects in training images, averaged over 6 runs. Variances are shown by the error bars

squares regression of the proposal accuracy $= 1 - (a + bn^{-2})$, $a = 0.619$, $b = 0.997$. Similar to the comparison for results of dataset 1, the comparison shows that these simple proposed relationships cannot fit the testing curves very well for dataset 2.

The training curves and testing curves in Figure 3.6 show the overfitting occurs in all the tests. We find that mean IoU for 3 of 1/25 images (containing 17 Gaussian objects) seems to be higher than 1 of 1/8 image (containing 24 Gaussian objects). Similarly, mean IoU for 8 of 1/25 images (containing 58 Gaussian objects) seems higher than 3 of 1/8 images (containing 69 Gaussian objects) as shown in Figure 3.5.

The mean IoU is calculated by averaging IoUs over all 5 classes, not demonstrating the model performance in details. Therefore we provide the curves of IoUs against the size of the training sets for each class as Figure 3.7 shows.

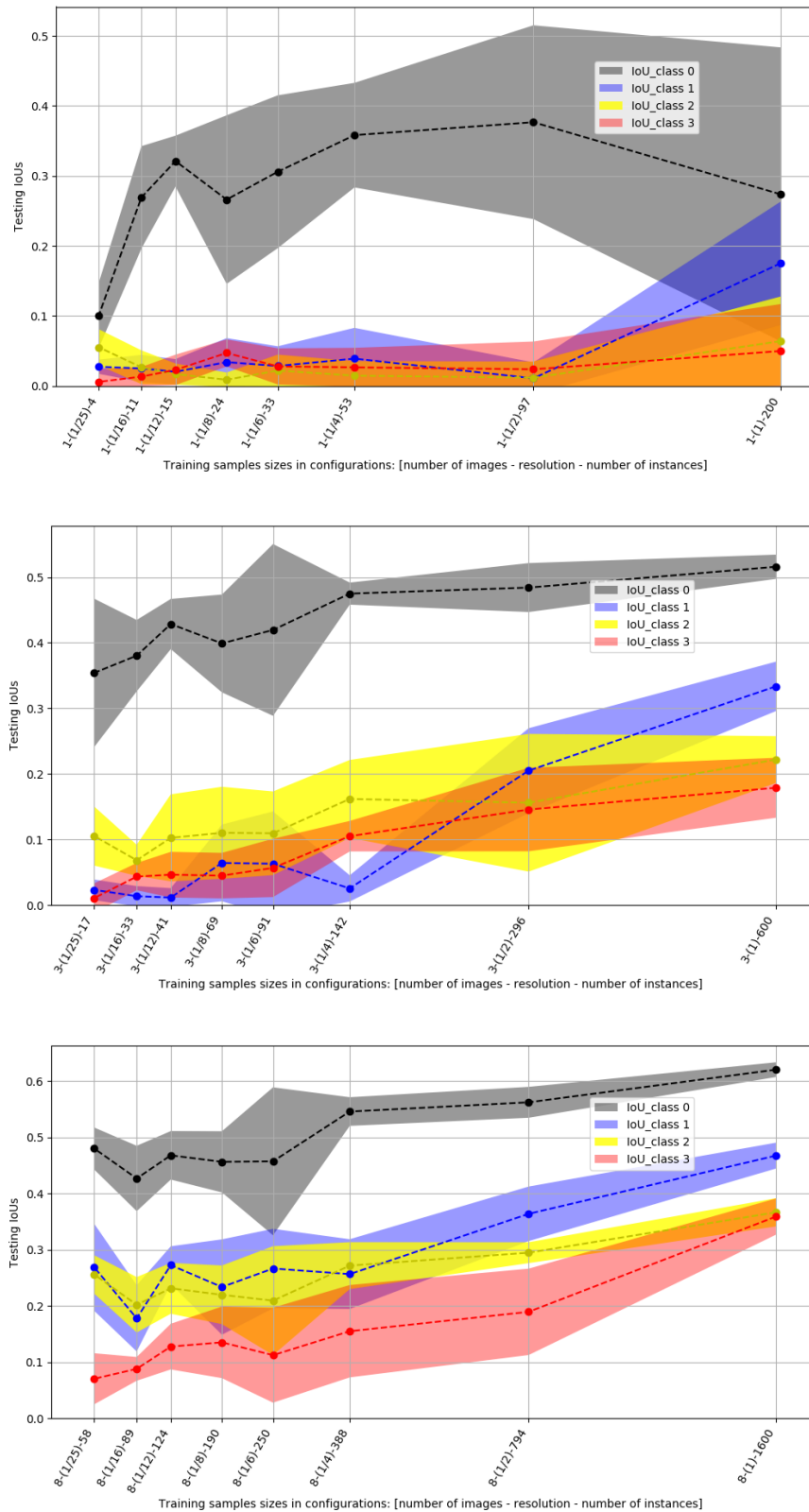


Figure 3.7: Testing results for [1 image], [3 images] [8 images] from top to bottom, showing IoUs of each class given the training sample sizes and how many objects in training images, averaged over 6 runs. Variances are shown as shaded

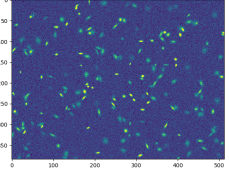
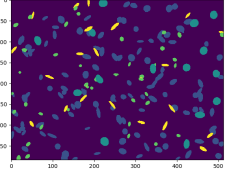
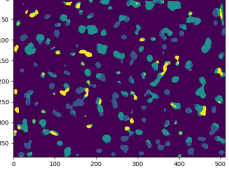
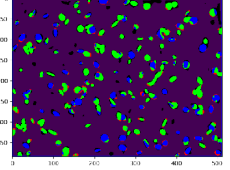
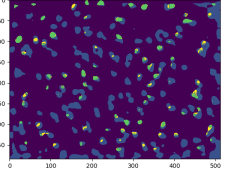
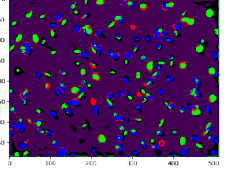
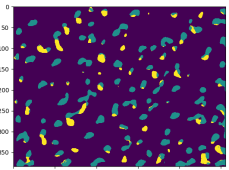
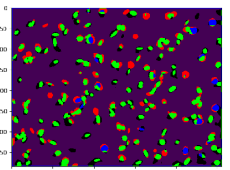
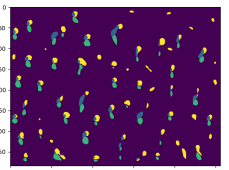
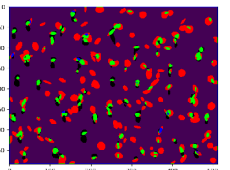
Testing image		Testing label	
			
Prediction	Overlay	Prediction	Overlay
			
1-[1-520x380]-200		1-[1/4-260x190]-53	
			
1-[1/16-130x95]-11		1-[1/25-104x76]-4	

Table 3.5: Example of the inference results of testing images for training sets group of [1 image], by models trained on in descending order of training size in configurations: [number of images - [part size - resolution] - number of instances]. In overlays, blue refers to correctly classified pixels, yellow refers to pixels of detected, type misclassified Gaussian objects, and red refers to pixels of undetected Gaussian objects

Table 3.5 shows some outputs of the model trained on the different-sized parts of 1 image, applied to the testing images. We observe that when the model is trained on this [1 image] group of the training set, the curve in Figure 3.7 shows that the IoUs of most of the classes except for class 0 are less than 20%, indicating a poor segmentation performance. In tests of all configurations of training parts of 1 image, overfitting occurs markedly. As Table 3.5 shows, training the model on a very small number of objects ends up with many pixels of undetected Gaussian objects (red in overlays) in the testing output. Overfitting on 1/25 of 1 image containing 4 Gaussian

objects, ends up with a broken object shape recognition. This [1 image] training set gives an evident example of the limited data issue as mentioned in Chapter 1: high sample bias, high outliers proportion, not representative for the data distribution.

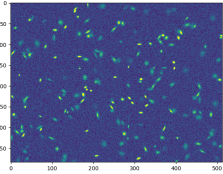
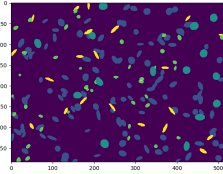
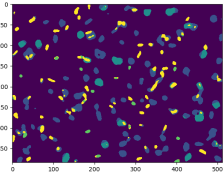
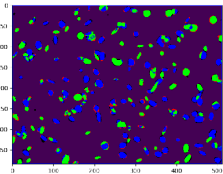
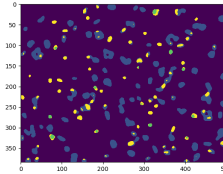
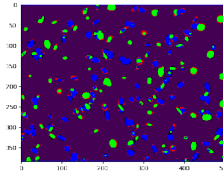
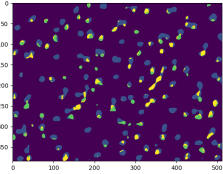
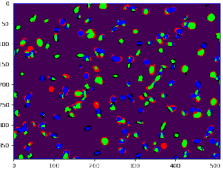
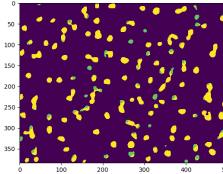
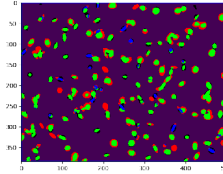
Testing image		Testing label	
			
Prediction	Overlay	Prediction	Overlay
			
3-(1-520x380)-600		3-(1/4-260x190)-142	
			
3-(1/16-130x95)-33		3-(1/25-104x76)-19	

Table 3.6: Example of the inferred results of testing images for training sets group of [3 images]. The format of the configuration and the overlays is as same as those of [1 image] above

Table 3.6 shows some outputs of the model trained on the different-sized parts of 3 images, applied to the testing images. In tests of all these configurations of training sets, the recognition does not break down. No prediction maps show many pixels of undetected Gaussian objects (red in overlays). However, the class balance of testing output deteriorates. As Figure 3.7 shows, the curve of each class crosses each other obviously.

Overfitting on some of the small size training sets results in well detected shape of Gaussian objects but with a broken class balance. The class balance in prediction

is labile as the objects in the training set are of imbalanced classes. For example, in Table 3.3, we can see that 1/4 of [3 images] training set containing 94 normal, 10 big, 21 small, 17 long class Gaussian objects. The IoU for class 1.big decreases from 20% to 2%, compared to it for 1/2 of [3 images]. Therefore the small fractions of data are likely to be unrepresentative for the distribution of the whole data with full images.

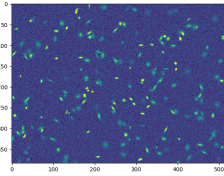
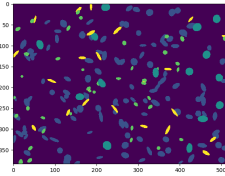
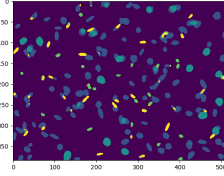
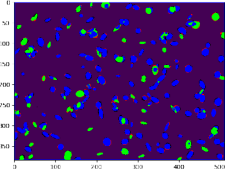
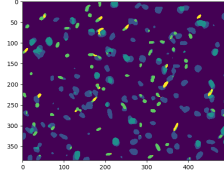
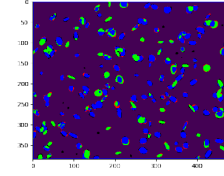
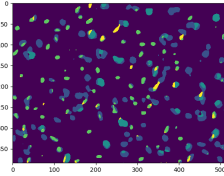
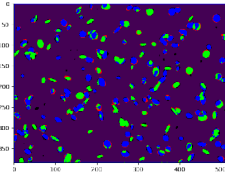
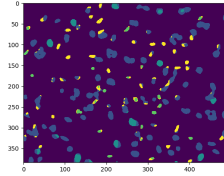
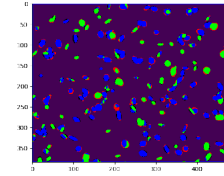
Testing image		Testing label	
			
Prediction	Overlay	Prediction	Overlay
			
1-[1-520x380]-1600		1-[1/4-260x190]-388	
			
1-[1/16-130x95]-89		1-[1/25-104x76]-58	

Table 3.7: Example of the inferred results of testing images for training sets group of [8 images]. The format of configuration and overlay are as same as those of [1 image] above

Table 3.7 shows some outputs of the model trained on the different-sized parts of 8 images, applied to the testing images. Objects recognition and segmented shapes are with relatively better performance compared to [3 images] training set. The accuracy is decreasing when training set size is decreasing. The decreasing is with a relatively more stable class balance as shown in Figure 3.7, compared to [3 images] training set.

In summary, there will be no certain small training set size making the recognition

broken. Instead, various representation of overfitting occurs. As mentioned before, a small training set is more likely to have issues of high sample bias, high outliers proportion compared to a large training set. In our experiment, we demonstrate that small size training data of our synthetic 2D-Gaussian image data may represent constrained objects distributions or imbalanced object classes.

We mentioned that the accuracy of training the model on 8 of 1/25 images (containing 58 Gaussian objects) is higher than 3 of 1/8 images (containing 69 Gaussian objects). By checking class distribution in training set in Table 3.3, we know that 8 of 1/25 images contain 31 normal, 8 big, 13 small, 6 long Gaussians, while 3 of 1/8 images contain 43 normal, 5 big, 10 small, 7 long Gaussians, Although 8 of 1/25 images contain less number of sample objects, the class is more balanced than 3 of 1/8 images.

Chapter 4

Approaches for semantic segmentation from limited labeled microscopic blood image dataset

4.1 Motivation

For our project, we have a dataset of microscopic blood images, while we don't have labels (annotations) for any type of objects such as red blood cells, white blood cells, platelets, etc. To train a deep network on the data for segmenting or counting, labels are necessary. For example, labeled data in the form of pixel-wise annotations are required to develop semantic segmentation. Counting objects in an image using deep regression networks requires position annotation often in the form of a dot at the centre of the object.

Human annotations are time consuming and sometimes inaccurate. Other limits such as lack of domain experts will make human labeling error prone, therefore it may be unlikely to obtain large amounts of well labeled data. In our case, we generate some labels for the red blood cells to have a set of labeled images. We propose approaches based on deep networks to train on only a small set of labeled images with an additional set of unlabeled images, and evaluate the methods on images with red blood cell annotations to see whether unlabeled images improve the model performance. For future analysis, we can apply our proposed method to other types of blood cells or platelets.

4.2 Related Work

Semi-supervised methods tackle the problem of limited training labels by making use of weak or unlabeled data for training. In the field of semi-supervised learning for classification, there are recent approaches showing high performance using fewer labels for MNIST classification.

One of the approaches that build an EM algorithm on a CNN is [59], getting

a 2.06% error rate using 300 labeled MNIST samples with 59k unlabeled samples. Compared to use only 300 labeled samples, the error rate decreased 6%. Their model is a 4 convolution layers CNN. Their method first learns the labeled data, followed by applying the model to unlabeled data to cluster it and pseudo-label it. A combined loss is defined by first a cross entropy between ground truth and prediction of real labeled data, and also another classification entropy for unlabeled data is added as an additional term. This is interpreted as an EM algorithm. They also have a confidence measure to acquire samples with high classification uncertainty. These are the samples that will be made pseudo-labels and added to the training set.

Another approach that uses both labeled and unlabeled samples is [4], reaching a 0.91 accuracy while the initial training accuracy with labeled data was at 0.84 for MNIST classification. They also train a CNN on some labeled data. Instead of a combined loss, they add the unlabeled data with the model's own prediction as pseudo-label, which is a self-training process. Their empirical results show that the classification accuracy increases the most 7%, comparing the accuracy after the self-training to the accuracy before the self-training.

For semantic segmentation, semi-supervised methods such as [53] shows that using a small number of pixel-level annotated images with a large number of weakly annotated images can obtain better performance compared to just using those pixel-level annotated images. In [53], they train two deep CNNs, where the first one is trained on images with pixel annotations, while the second one is trained on other images with the weak annotations such as the bounding boxes. The second CNN generates some bias for all object classes. These biases are used to boost present foreground classes more than the background class. The biases are involved in the loss of the first CNN, therefore their model proposed can train from with both images with strong labels and images with weak labels. They have a testing IoU of 67.6% by training their model on all 10k images with pixel-level annotations. The combined datasets of 1.4k pixel-level labeled images and 9k bounding box labeled images achieve a testing IoU of 64.6%. Training the model on only 1.4k pixel-level labeled images results in a testing IoU of 62.5%. Their semi-supervised model increases the mean IoU by 2% for semantic segmentation validated on PASCAL VOC 2012 dataset.

4.3 Self-training (Bootstrapping) method

Bootstrapping, or self-training, is one of the semi-supervised methods that start with training on a small quantity of labeled data and continues the training, adding unlabeled data into the training set with its own prediction as ground truth. Self-training is an unsupervised scheme that makes it possible to continue a learning process on unlabeled data to help increase performance. Inspired by these previous works in Chapter 4.2, we investigate an EM-like self-training method for semantic segmentation with the purpose of counting. In our case, this begins with a supervised stage, in which the model is initially trained on limited labeled data. An unsupervised stage follows that, where the model predicts labels for the next batch of (unlabeled) data and uses its own predictions as pseudo-labels. The prediction is the probability map provided by the last sigmoid layer. The question we investigate here is how we should convert these probability maps into labels. In particular, we would like the network to train only on those pixels for whose predictions the network is sufficiently confident. For this reason, we need a confidence measure for thresholding. To measure the prediction confidence, Monte Carlo (MC) Dropout [21] can be used to calculate pixel-wise image uncertainty estimation by computing the mean and variance of T forward passes through the network predicting the same pixel with dropout, thus obtaining an uncertainty map the same size as the output. The following methods are compared to generate pseudo labels.

1. *Mean probability*

The mean (overlined) of the T Monte Carlo samples of the sigmoid layer output probabilities (pb_1, \dots, pb_T) (i.e. a real value in $[0, 1]$) will be added into the training set as the label (L) for that pixel (p).

$$L(p) = \overline{pb_1, \dots, pb_T} \quad (4.1)$$

2. *Mean binary threshold*

For each pixel label $L(p)$, we consider its mean probability value over T Monte Carlo samples. If the mean is higher than a threshold τ , then this pseudo-label

is set to 1, otherwise it is set to 0.

$$L(p) = \begin{cases} 1, & \text{if } \overline{pb_1, \dots, pb_T} > \tau \\ 0, & \text{otherwise} \end{cases} \quad (4.2)$$

3. *Mean probability threshold*

For each pixel label $L(p)$, we consider its mean probability value over T Monte Carlo samples. If the mean is higher than a threshold τ , then this pseudo-label is set to 1, otherwise it is set to the mean probability value.

$$L(p) = \begin{cases} 1, & \text{if } \overline{pb_1, \dots, pb_T} > \tau \\ \overline{pb_1, \dots, pb_T}, & \text{otherwise} \end{cases} \quad (4.3)$$

4. *Nearby pixels threshold*

The label $L(p)$ of a pixel depends on the sum of the sigmoid probabilities of the 3×3 pixel patch ($pt_{3 \times 3}$) that contains the pixel under consideration. For each of the T Monte Carlo samples, a threshold τ is compared against the sum of the predicted values of the pixel patch. If the sum is greater than τ , the label of the pixel is set to 1. If the sum is smaller for all T samples, the pseudo-label is set to the mean probability value.

$$L(p) = \begin{cases} 1, & \text{if any } t \in T \quad \sum pb_t(pt_{3 \times 3}) > \tau \\ \overline{pb_1, \dots, pb_T}, & \text{otherwise} \end{cases} \quad (4.4)$$

In addition to measuring the confidence of predictions when generating self-supervised labels, we make the loss function in our approach a weighted pixel-wise cross entropy (CE), which utilizes the variance of T predictions to help the classifier assess the trustworthiness of the label of a pixel.

$$Loss = \sum_{i,j} \frac{1}{\widetilde{\sigma_{ij}^2}} * CE(y_{p_{ij}}, \hat{y}_{p_{ij}}) \quad (4.5)$$

Let i, j denote the coordinates of an image pixel. Further, $\hat{y}_{p_{ij}}$ is the predicted class of the pixel and $y_{p_{ij}}$ is the true label of the pixel. For each pixel, $\widetilde{\sigma_{ij}^2}$ denotes the variance of its T predicted probabilities normalized (min-max scaled) to interval

[1,100] over all variances of pixels in this image. The weight we use is the reciprocal of the $\widetilde{\sigma_{ij}^2}$. Therefore, pixels with high variance predictions are weighted less during the training, reducing the effect of learning from pixels with uncertain predictions.

Our self-training approach is based on segmentation with U-Net, which is a popular encoder-decoder network for semantic segmentation in the medical domain. It has down-sampling layers and up-sampling layers forming a u-shaped architecture. In particular, U-Net has shown excellent performance on cell tracking in biomedical image segmentation tasks. It was one of the winners of the ISBI cell tracking challenge in 2015. Figure 4.1 shows the work-flow of our approach.

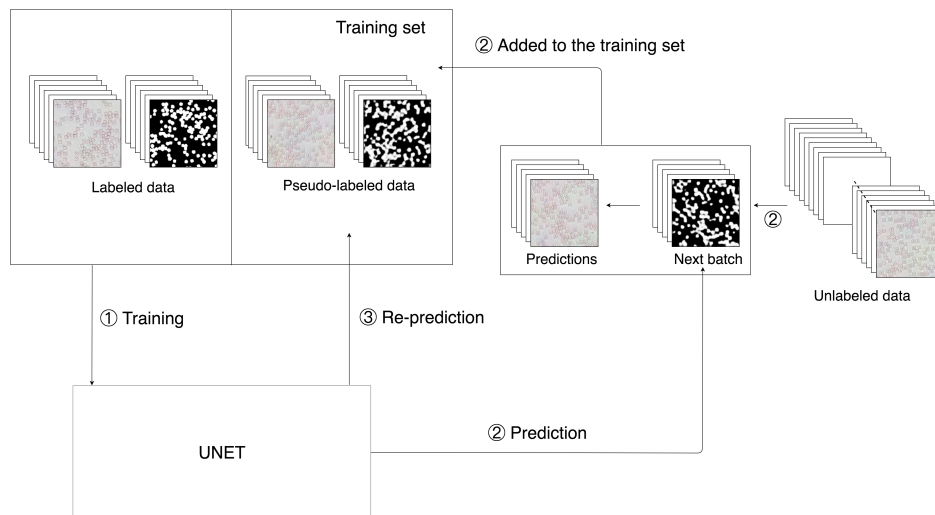


Figure 4.1: The work-flow of our self-training model based on U-Net

Our approach starts with training the U-Net on a very small number of images with annotations. We pretend to not have annotations for the rest of the images to simulate the scenario in which we lack ground truth. After the initial supervised stage, the self-training model makes predictions on the next batch of unlabeled data and generates pseudo labels using the various confidence threshold methods. The pseudo-labeled data is added into the training set and the model is re-trained. This is repeated for multiple iterations.

4.3.1 Results on semantic segmentation

The experiments are done with a human blood sample taken with a lensless microscope from Alentic Microscience Inc. [1]. The dataset contains 600 image samples,

where each image has a fixed size of 128×128 RGB pixels containing more than 150 cells which are not superimposed over each other. The pixel-wise annotations are binary masks of two classes: red blood cells, and others. We split these data into 504 training and 112 test samples. Our self-training model is based on an implementation with Keras of the U-Net architecture [24]. The self-training approach will not work if the initial accuracy is not sufficiently high because it makes the model more likely to amplify errors. On the other hand, if the initial training set is large enough to yield high accuracy, there will likely be little improvement with self-training. The test results of segmentation pixel accuracies and Intersection over Union (IoU) averaged over 5 runs for 300 iterations with different initial training sizes are shown in Figure 4.2. We found that training on *one* 128×128 image gives sufficient accuracy for the rest of the method to work well.

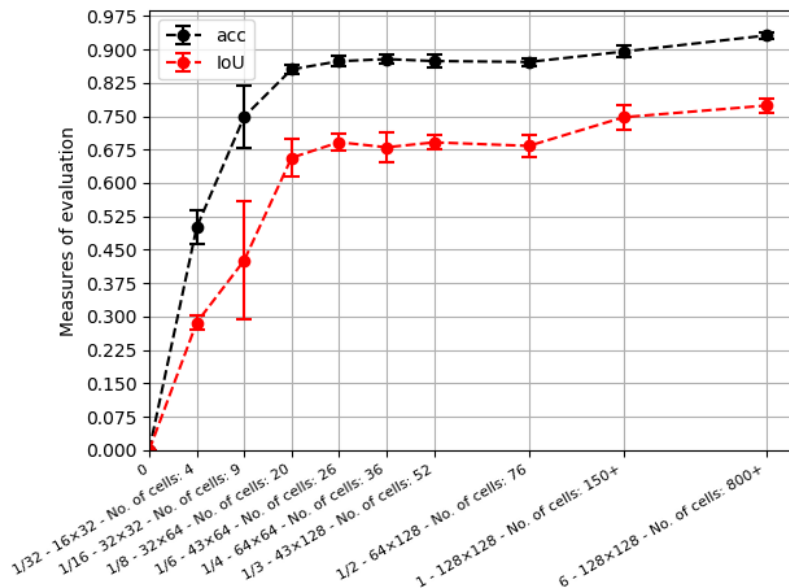


Figure 4.2: Average testing accuracies and IoUs for training the network with different initial training sizes in configurations given by: [percent of a full image, pixels, number of objects]

We tested all methods for generating pseudo-labels for predicting unlabeled data. We use $T = 100$ forward passes and a dropout rate of 50% for MC dropout inference on unlabeled data. For each iteration, the model is trained on the current training set and predicts the labels of the next *one* unlabeled image. In each iteration, we train 100 epochs on the current training set. The results are presented in Table 4.1, including

testing accuracy(acc), Intersection over Union(IoU), positive predictive value (PPV), true positive rate (TP), true negative rate (TN) and the positive prediction rate (P) averaged over 5 runs.

Threshold	iters	Measures					
		acc	IoU	PPV	TP	TN	P
Mean probability	15	0.889→0.720 (±0.010)(±0.017)	0.734→0.349 (±0.020)(±0.029)	0.886→0.971 (±0.008)(±0.006)	0.814→0.269 (±0.021)(±0.045)	0.936→0.995 (±0.004)(±0.001)	0.349→0.105 (±0.007)(±0.017)
Mean binary threshold	28	0.889→0.851 (±0.013)(±0.020)	0.735→0.621 (±0.029)(±0.052)	0.880→0.970 (±0.011)(±0.005)	0.818→0.628 (±0.028)(±0.056)	0.932→0.988 (±0.006)(±0.003)	0.353→0.246 (±0.009)(±0.023)
Mean probability threshold (0.5)	28	0.885→0.910 (±0.008)(±0.014)	0.723→0.759 (±0.021)(±0.091)	0.889→0.885 (±0.007)(±0.024)	0.796→0.878 (±0.022)(±0.056)	0.939→0.929 (±0.005)(±0.019)	0.334→0.377 (±0.010)(±0.032)
Mean probability threshold (0.55)	28	0.887→0.902 (±0.015)(±0.014)	0.728→0.764 (±0.034)(±0.030)	0.885→0.880 (±0.014)(±0.021)	0.807→0.858 (±0.035)(±0.020)	0.936→0.928 (±0.008)(±0.013)	0.346→0.370 (±0.013)(±0.008)
Nearby pixels threshold (8)	15	0.889→0.745 (±0.007)(±0.023)	0.732→0.389 (±0.015)(±0.045)	0.889→0.974 (±0.023)(±0.004)	0.809→0.338 (±0.026)(±0.063)	0.938→0.994 (±0.016)(±0.002)	0.346→0.132 (±0.018)(±0.025)

Table 4.1: Summary of mean and variance of segmentation accuracies, IoU, PPV, TP, TN, P over 5 runs. Left: Performance measures in evaluation after training on labeled data (before self-training). Right: After self-training.

For using *mean probabilities* as pseudo-labels, the initial accuracy and IoU were 88.9% and 73.4%. After 15 iterations with 15 images with pseudo-labels added into the training set, the accuracy and IoU decrease to 72.0% and 34.9%, and the positive prediction rate decreases from 34.9% to 10.5%. The predicted values of positive predicted pixels become smaller with an increase in PPV of 97.06%. TN would be high when most of the predictions are 0s.

In our tests with pseudo-labels using a *mean binary threshold*, the accuracy decreases slightly from 88.9% to 85.1%. The IoU and TP become lower from 73.5% to 62.1%, 81.8% to 62.8% respectively. Since our threshold for the pseudo-labels makes the pseudo-labels binarized. This results in the fact that the predicted labels of the model trained with these pseudo-labels will be produced close to 0 or 1. This leads to an increase in PPV of 97.0%. As Figure 4.3 shows, cells become thinner, which shows a result of under-segmentation.

Reaching a slight improvement of accuracy and IoU can be achieved by using *mean probability threshold* pseudo-labels. Pixel accuracy increases 1-2% and IoU increases 3-4%. This threshold scheme guides the network to emphasize confident high-value pixels, making the network make 3-4% more positive predictions and a 5-7% more true positive predictions. The TN rate decreases very slightly around 1%. More pixels of cell walls are predicted as positive as shown in Figure 4.3.

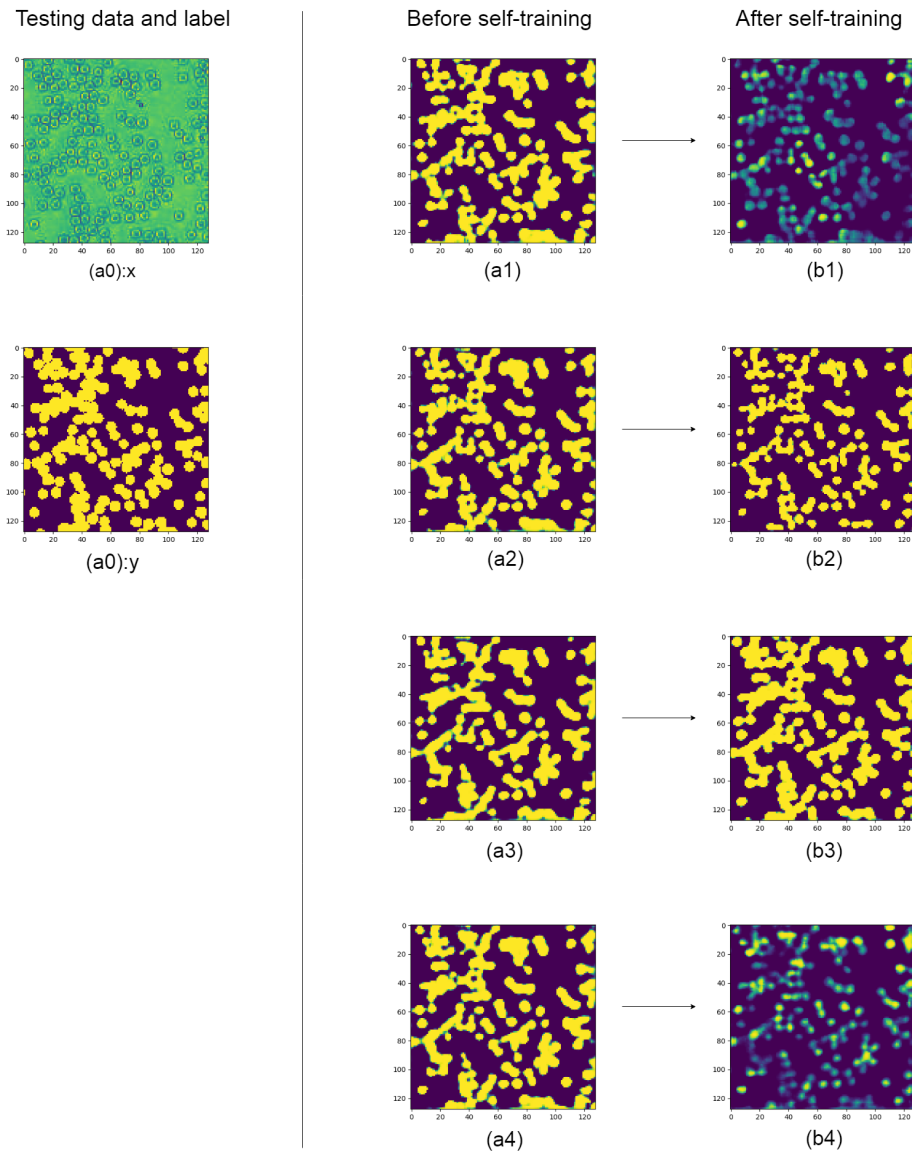


Figure 4.3: Inference results: (a0)x, (a0)y: Input data and label. (a1)-(a4): After training on labeled data. (b1): After 15 self-training iterations(ends up with training on 1 labeled data + 15 pseudo-labeled data) using mean probability pseudo-labels. (b2): After 28 self-training iterations using mean binary threshold (0.5) pseudo-labels. (b3): After 28 self-training iterations using mean probability threshold (0.5) pseudo-labels. (b4): After 15 self-training iterations using nearby pixels threshold (8) pseudo-labels.

For *nearby pixels threshold* pseudo-label, a threshold value of 8 means that when creating the pseudo-label, a pixel is set to 1 only if the predicted values of the sum of a 3×3 pixel patch containing current pixel is greater than 8. This results in a very low TP of 33.8%. When the model is trained on *Nearby pixels threshold* pseudo-labels,

the speed of decrease in the value of different predicted pixels varies remarkably. In particular, the predicted value of pixels near cell walls change rapidly, a fact that we can use later for our counting algorithm.

4.4 Counting algorithm

While the self-training approach did not improve much the results of segmentation, we noticed a slight image level improvement on the shape mask of cells. In particular, the cells' boundaries become somewhat clearer after applying self-training with *nearby pixels threshold*. Therefore, we propose here a counting method based on these pixel value changes to test this hypothesis. This algorithm is outlined below.

Algorithm 1 Using output sigmoid map of self-training for counting

Require: Segmentation sigmoid output map after self-training

```

1: procedure SUB_MAPS( $map, c1, c2, n$ )                                ▷ Subtraction between a
   Laplacian filtered map of pixels less than a constant  $c1$  and a map of Laplacian
   filtered pixels greater than a constant  $c2$ 
2:    $map1 \leftarrow NORM(LF(map)) < c1$ 
3:    $map2 \leftarrow NORM(LF(map)) > c2$ 
4:                                                                      ▷ LF: Laplacian_filter
5:                                                                      ▷ NORM: Standard_scaler [54]
6:    $diff \leftarrow map1 - DILATED(map2)$ 
7:    $dot\_pred \leftarrow CLEAN(diff, n)$ 
8:   return  $dot\_pred$                                                 ▷ the dots prediction
9: end procedure
10:
```

We first apply a Laplacian filter: a 2-D isotropic measure of the 2nd spatial derivative of an image [34], on the predicted map (M) of the self-trained model to get a map (∇M) indicating rapid changes. An image is a matrix of discrete values so the derivatives are approximate by convolving an image with a kernel $[[0,1,0],[1,-4,1],[0,1,0]]$ in OpenCV [6]. The values of the ∇M are normalized to remove the mean and scaled to unit variance. We choose two thresholds $c1$ and $c2$. Let $map1$ be a

```

11: function CLEAN(map,n)
12:   for all pixel  $\in$  map do
13:     current_window  $\leftarrow$  GET_AREA(pixel, map, n)
14:                                      $\triangleright$  get a  $n \times n$  window of map
15:     if SUM(current_window)  $>$  1 then
16:       current_pixel  $\leftarrow$  1
17:       current_windows  $\leftarrow$  0 for other pixels
18:     end if
19:   end for
20:   return map
21: end function

```

binary map with 1's only at those pixels where ∇M had a value less than $c1$. *map1* corresponds to the regions where the change in M was small. Similarly, let *map2* be a binary map with 1's only at those pixels where ∇M has a value larger than $c2$. We subtract *map2* from a *map1* to get our pixels of interest (*diff*): isolated pixel components corresponding to individual cells. Since *map1* and *map2* do not overlap, *map2* is dilated by a 2×2 kernel before subtraction. The resulting image (*diff*) is then cleaned by being scanned by a $n \times n$ window, replacing small non-connected pixel components with a single dot denoting a cell.

We conduct experiments on our microscopic cell dataset and a synthetic 2D-Gaussian dataset generated by ourselves. In our test with cell images, the results shown here are averages over 5 runs for 50 testing images (128×128) with manual dot annotations where each dot corresponds to one cell. Since the predicted dots are not necessary to be the centre of the cells, we evaluate the dot prediction by doing an injective function. Each dot in the prediction map is mapped to at most one dot in the ground truth map with a restriction of a limited distance (≤ 7 Euclidean distance of pixels). The dots that end up being not mapped in predictions are labeled as FP, while those in the ground truth are FN.

In Table 4.2, the results of counting cells images are presented. A higher threshold value $c2$ increases the number of P(positive prediction), leading to a higher TPR but lower PPV. A lower threshold value $c1$ does the opposite. Figure 4.4 shows an example

method	c1	c2	evaluation measure						
			P	GT	TP	FP	FN	TPR	PPV
our method	-1.00	0.45	140.9 (± 9.6)	144.7 (± 10.2)	131.8 (± 9.1)	9.0 (± 3.3)	12.8 (± 4.5)	0.911	0.936
	-1.00	0.85	147.3 (± 9.9)	144.7 (± 10.2)	134.8 (± 9.6)	12.5 (± 3.6)	9.9 (± 3.5)	0.932	0.915
	-1.45	0.45	130.7 (± 8.5)	144.7 (± 10.2)	125.2 (± 8.3)	5.4 (± 2.4)	19.4 (± 5.7)	0.866	0.958
Circle Hough transform			132.0 (± 9.6)	144.7 (± 10.2)	122.8 (± 8.5)	9.3 (± 3.4)	21.8 (± 5.7)	0.849	0.930

Table 4.2: Evaluation of cell counting on 50 testing images by three pairs of $(c1, c2)$, iterations of self-training=7, $n=5$. The numbers in columns: P(positive), GT(ground truth), TP(true positive), FP(false positive), FN(false negative) correspond to mean and standard deviation of the number of cells, in columns: TPR(true positive rate), PPV(positive predictive value) correspond to rates calculated from the whole 50 samples.

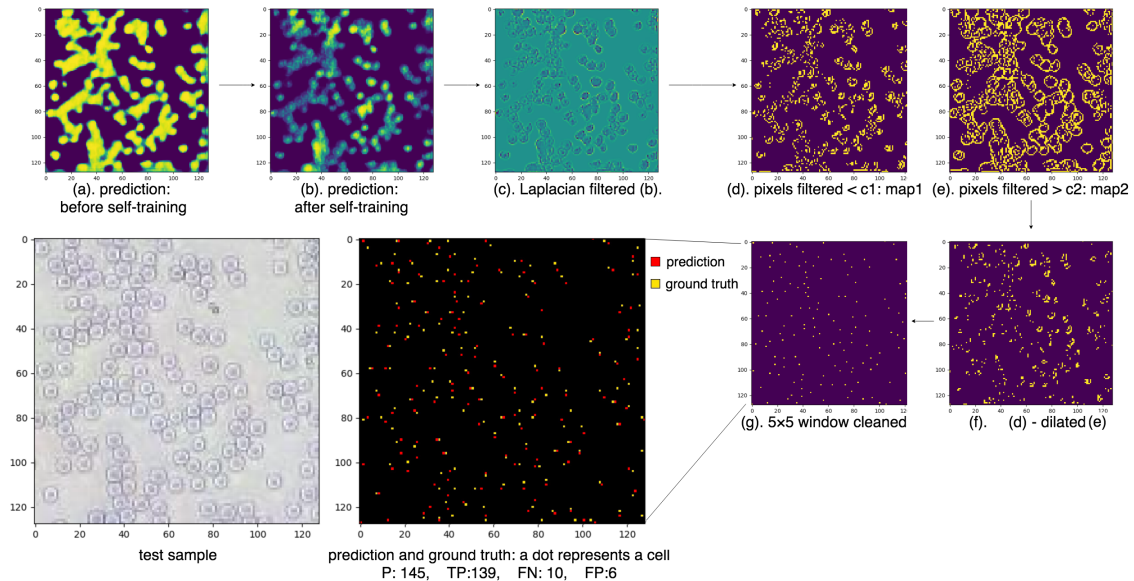


Figure 4.4: A test sample for the counting process on cells images: (a): Prediction after training on 1 labeled data. (b): Prediction after 7 self-training iterations (1 labeled data + 7 pseudo-labeled data. Pseudo label: Nearby pixels threshold). (c): Laplacian filtered(b). (d): $map1$ in Alg 1. (e): $map2$ in Alg 1. (f): $diff$ in Alg 1. (g): Dots prediction

of the counting process on a test sample. This algorithm is one of the possible ways to utilize the output map of self-training by creating thresholds. Algorithms such as the Circle Hough Transform(CHT) algorithm [2] or watershed segmentation [55]

could also be used to process output segmentation map after self-training to count cells.

CHT is an image processing algorithm to detect circles, and we use this algorithm here to compare it to our algorithm although our algorithm is more general as it can detect different shapes. The CHT algorithm outputs a dot annotation same size as the input image where each dot is the centre of a circle detection. The minimum search radius is set to 4 and the maximum is set to 5 pixels. We evaluate the detection performance using the same method as stated above. The result of using CHT to count cells is shown in Table 4.2. Our algorithm outperforms CHT by a 7% more TPR with c1 of -1 and c2 of 0.45.

The other dataset we used to test our algorithm is a synthetic dataset where we generate 200 2D-Gaussian objects at random positions in an image with Gaussian noisy as background. The standard deviations of those 2D-Gaussian objects are in range 1 to 5 pixels. We save the coordinate of the mean(centre) of each Gaussian object after they are generated. The main difference between this dataset and the cell dataset is the Gaussian objects may overlap one another. We repeated the same experiment on this dataset as on the cell data. That is, we train the UNet on one image of this 2D-Gaussian image with annotation, followed by self-training using *nearby pixels threshold*. The results of applying our counting algorithm to the segmentation map of those 2D-Gaussian images are presented in Table 4.3. Since it's more challenging to count objects that may overlap, the TPR is 1-2% lower and the PPV is 2-3% lower than results for cells images. Figure 4.5 shows an example of result for counting on a 2D-Gaussian image.

c1	c2	evaluation measure						
		P	GT	TP	FP	FN	TPR	PPV
-2.00	1.00	196.6 (±9.3)	200.0 (±0.0)	180.3 (±5.0)	16.3 (±5.7)	19.7 (±5.0)	0.902	0.917
-2.00	2.00	209.6 (±11.2)	200 (0.0±)	183.0 (±5.0)	26.6 (8.1±)	17.0 (±5.0)	0.915	0.873

Table 4.3: Evaluation of counting on 10 testing images of synthetic 2D-Gaussian by two pairs of (c1,c2), iterations of self-training=8, n=6.

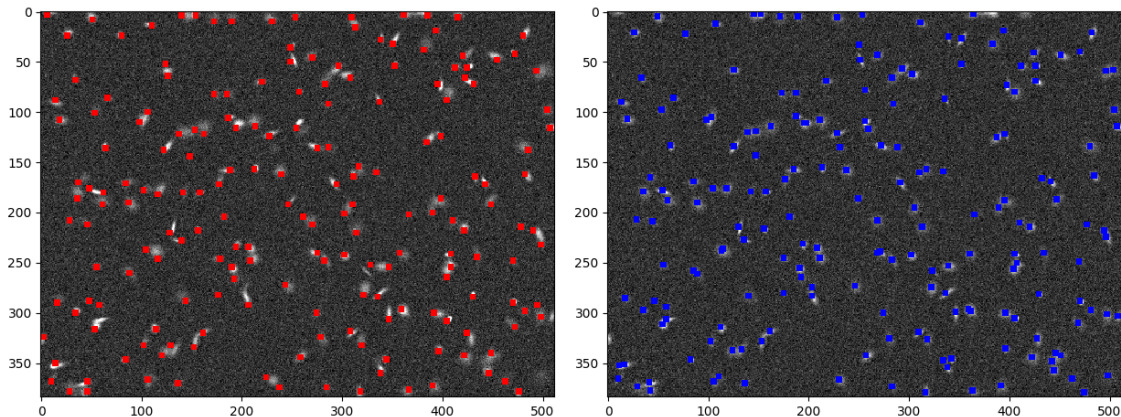


Figure 4.5: A test sample for the counting on 2D-Gaussian synthetic images. Left: predictions. Right: Ground truth (centres of each 2D-Gaussian objects). P:192, TP:183, FP:9, FN:17.

4.5 Discussion

While we proposed this self-training method originally for improving segmentation accuracies for datasets with small numbers of examples using self-training, we found that results critically depend on the thresholding method, and we only found a small improvement in that regard for one of the methods proposed. Interestingly, we found that visually modifying the data based on the pseudo-labels helped with the identification of cell boundaries. This in turn helped with the identification of cells and, ultimately, with the corresponding counting. This is a promising new direction for using model predictions in combinations with uncertainty measure. We hope that by using a machine learning approach, we can first match results on the easier tasks, and then will have a better chance of generalizing to the ones that are still considered very hard.

Counting red blood cells is possible to solve with traditional Computer Vision(CV) techniques, but those techniques such as color or shape detection do not generalize to harder visual contexts. This counting method may be useful when it's hard to extract objects of interests since this method works on the predicted segmentation map. We hope to try more for generalizing the contexts considered harder: other datasets containing more classes and a variety of shapes and sizes of objects. It would also be worth trying to make the pseudo-label generation be dynamic based on scenes or classes.

4.5.1 Active learning

Active learning is another learning method that we consider combining into our self-training application. Active learning is the learning method involving human users during the training. These methods are proposed for human users/teachers to have more control over the training process. With regard to the limited labeled data problem, if the assumed scenario is that collecting unlabeled data is not high-cost, while labeling is high-cost. In such a scenario, active learning algorithms can reduce the cost by interactively querying the user for doing online labeling.

During this labeling process, the user can choose some targeted samples (unlabeled images) to label, which means some manual selection. The selection is for adding informative training samples, avoiding adding abundant training samples. Therefore, the user will seek the least but sufficient amount of labeled data required to have a certain desired performance. There are two parts of an active learning method is important: query strategy and interaction tool.

The query strategy is various and essential for active learning algorithms. The query strategy can be selecting the more representative unlabeled samples by picking the most uncertainty samples. This strategy is based on an uncertainty measure. Another query strategy can be selecting those unlabeled samples, where the prediction of the model made most errors. With this strategy, the active process aims at error reduction and better generalization.

The interaction tools can be either self-made labeling tool or outsource labeling applications. For example, we made a dot annotation labeling tool used in our counting experiment as Figure 4.6 shows. If we train a deep network such as fully regression convolutional networks [75] for counting objects. We can actively create dot annotations for images, reaching the number of labeled images needed for an expected performance. In addition to self-made labeling tools, outsource labeling applications can be used, such as open source tools mentioned in Chapter 2.5.1. Using some well-built open source labeling applications is time-saving and reliable. The interaction tool can be embedded into the training program, being opened when querying the user. This kind of embedding methods cost more memories. To save memories, another way is when the query happened, the model stops, saves all the weights and call the interaction tool. After the user finished actions, the model reloads all the

weights and continue the training.

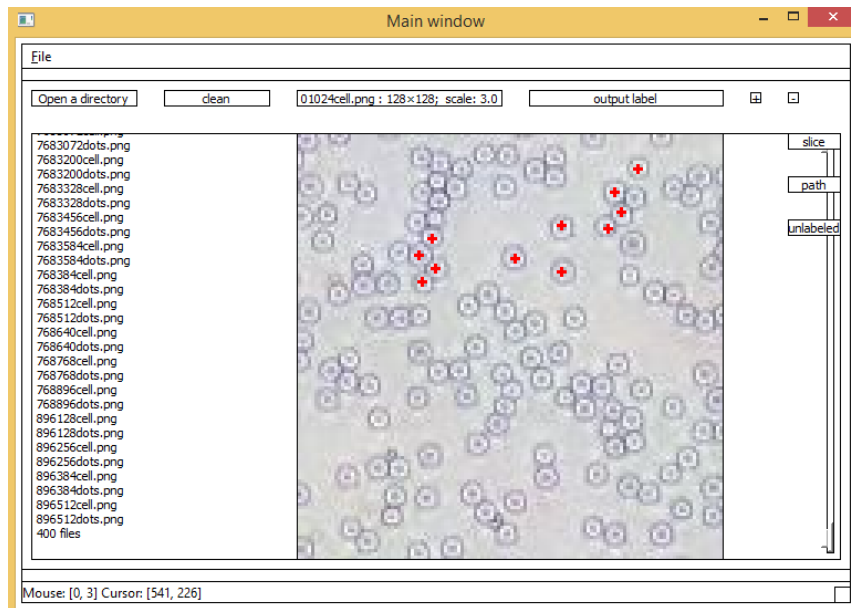


Figure 4.6: A self-made dot annotation labeling tool

In our experiments of doing semantic segmentation on cell images, active learning is not used during the training. The labeling tool we built is for testing the counting algorithm. The idea of combining self-training and active learning is worthwhile to be discussed. One good entry point for utilizing active learning is to fix the mis-labeled pixels in pseudo-labels manually and if the pseudo-labels generated deteriorates during the self-training process, human interaction can stop it and adjust the hyperparameters.

Another work that is worth trying is to make a secondary structure for self-training. One possible method aimed at achieving a better performance, is that using dot annotated predictions as a second type pseudo-labels, where we can manually correct the errors using our dot annotation tool easily. Dot annotations can be added into the training set as the object-level labels to influence the segmentation and ultimately improve the accuracy of both the segmentation and the counting. The labels for training a semantic segmentation are the pixel level annotations that do not distinguish objects. If we use the dot annotations to indicate the individual objects. The object-level labels can improve the model performance by constructing some frameworks trained on both pixel-level labels and object-level labels.

Chapter 5

Conclusion

In this section, we present a summary of the results of all our experiments, and provide potential future work.

5.1 Summary of results

In the thesis, the results of our breakdown experiment show that when the model is trained on an extremely small training set, there is no certain breakdown line for the recognition while remarked overfitting occurs. The overfitting makes the model generalize monotonously leading to a low testing accuracy. The results suggest that creating and labeling some representative data is more important than enlarging the size of the dataset.

The results of our self-training method doing semantic segmentation from microscopic limited labeled blood images show that choosing the type of pseudo-label is essential. The performance of the self-trained model critically depends on the thresholding method related to uncertainty measure inside the pseudo-label generation function. We only found a small improvement in that regard for one of the thresholding method.

The results of our counting method which works on the segmentation output show that there are 6-9% errors for our blood cell dataset and synthetic 2D-Gaussian dataset. The algorithm needs to be improved and there is some space to increase the accuracy of counting. Since our algorithm is based on two thresholdings of the changes in pixel values, the thresholding scheme to generate the non-connected components representing objects can be improved by a better normalization of all pixel values. Also, the scan method replacing each of the non-connected components with a dot label can be improved by adding some condition check. The check can focus on, for example, whether a long or big connected component is multiple objects or whether a small area containing multiple non-connected components is a single object.

5.2 Future work

The proposed self-training method is a first attempt for combining the idea of bootstrapping into semantic segmentation. The discussion on adding active learning functions to improve the model performance has been provided. To do this, we will start by building the interactive function that can query the user during the self-training process. The query strategy could be error reduction oriented, with human fixing the error of pseudo-label manually. For each time the model picks some of the unlabeled images and predicts on those images, the prediction, the overlay, and the MC confidence heatmap will be shown to users through a user interface. The user interface embeds some labeling tools with some functions such as highlighting the object shape’s outline, to help the user fix the pixels of the edges of an object. The user can fix the errors by observing the overlay and also the MC confidence heatmap to see the pixels that the deep network are not confident to predict on.

Since the self-training is interpreted as an Expectation Maximization, the user can guide the E step by interrupting and tuning the parameters in the pseudo-label generation. For example, the user can change the thresholding constant during the training process. If the user see more and more pixels of the edges of an object are classified as part of the object by the model with an increasing confidence, while the user’s expectation for the segmentation is that a low false positive must be guaranteed, which means the model should make the least pixels not part of the object to be mis-classified as part of the object. The user can increase the value of thresholding constants to guide the model to be more strict to classify a pixel as part of the object. The interactive functions are where the active idea combined.

Recently, some hybrid methods combining the sub-structure of the one-shot learning model, Regional CNN proposal model, demonstrates doing fast object tracking and semi-supervised object segmentation in real time [72]. We can combine similar ideas that can accelerate the model. One idea is to make user label just object-level of annotation such as bounding box or landmark instead of pixel-level annotation. A specific pretrained network such as a mask-RCNN [29] is responsible for detecting the shape inside the bounding box or landmark. Another idea is once the first frame of a video is labeled with bounding boxes, the latter frames are predicted by the model to generate more training data. We would like to explore approaches doing semantic

segmentation for real-time video frames data with limited labels.

Appendices

Appendix A

Dataset sizes in papers of deep learning models for semantic segmentation

Architecture	Benchmark datasets			
	VOC	COCO	Cityscapes	other
FCN	VOC 2010: T:8494			NYUDv2 T: 795 E: 654SIFT Flow: T: 2,488 E: 20033 classes
Segnet	PASCAL- Context:VOC 2010: 59 classes			CamVid road scenes:T: 367 E: 233SUN RGB-D: T: 5285 E: 5050
Unet				VNC: T:30
RefineNet	VOC2012: T:1464, V:1449, E:1456		T:2975 E:500.	Person-Part: T: 1717, E: 1818NYUDv2 T: 795 E: 654SUN RGB-D: T: 5285 E: 5050ADE20K MIT: 20K
DeepLab v1	PASCAL- Context:VOC 2010:59 classes T: 4998 E: 5105 VOC2012: T:1464, V:1449, E:1456			
DeepLab v2	VOC2012:T:1464, V:1449, E:1456PASCAL- Context:VOC 2010: 59 classes T: 4998 E: 5105 Person-Part: T: 1716 E:1817		T:2975 E:500V: 1525	
DeepLab v3	VOC2012: T:1464, V:1449, E:1456		T:2975 E:500V: 1525	COCO: images that have annotation regions larger than 1000 pixels and contain the classes defined in PASCAL VOC 2012

PSPNet	VOC2012: T:10,582(augmented), V:1449, E:1456	T:2975 E:500V: 1525	ADE20K: 150 classes, 1,038 image-level labels.
Large Kernel Matters	VOC2012: T:1464, V:1449, E:1456		Semantic Boundaries Dataset T: 10,582
CRFasRNN	A Customized VOC2012:total of 11,685 images PASCAL-Context: VOC 2010:T:10,103 E:9,637 images.	MS COCO 2014: 66,099	
DeepMask	PASCAL VOC 2007: 9,96320 classes	MS COCO 2014: T:80,000 E:5000	
2D-LSTM			Stanford Background dataset:T: 572 E: 143 8 classesSIFT Flow: T: 2,488 E: 20033 classes
R-CNN	PASCAL VOC 2007: 9,96320 classesVOC2012: T:1464, V:1449, E:1456		ILSVRC2012(auxiliary):1000 categories and 1.2 million images
Fast R-CNN	VOC 2007 + VOC 2012:augmented 16.5k in total PASCAL VOC 2007: 9,96320	MS COCO:T: 80k	
Faster R-CNN	classesVOC2012: T:1464, V:1449, E:1456	MS COCOT: 80k V:40kE:20k	
Mask R-CNN		MS COCO:T: 80k + 35kE: 5k	T:2975 E:500V: 1525

Table A.1: Summary of the dataset and the number of data used for influential deep learning semantic segmentation networks. Samples are denoted as T : training, E: Evaluation(validation or testing).

Bibliography

- [1] Alentic Microscience Inc. <http://www.alenticmicroscience.com/>, 2019.
- [2] Tim J Atherton and Darren J Kerbyson. Size invariant circle detection. *Image and Vision computing*, 17(11):795–803, 1999.
- [3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.
- [4] Dor Bank, Daniel Greenfeld, and Gal Hyams. Improved training for self training by confidence assessments. In *Science and Information Conference*, pages 163–173. Springer, 2018.
- [5] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.
- [6] G. Bradski. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*, 2000.
- [7] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.
- [8] Gabriel J Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *European conference on computer vision*, pages 44–57. Springer, 2008.
- [9] Albert Cardona, Stephan Saalfeld, Stephan Preibisch, Benjamin Schmid, Anchi Cheng, Jim Pulokas, Pavel Tomancak, and Volker Hartenstein. An integrated micro-and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy. *PLoS biology*, 8(10):e1000502, 2010.
- [10] Joo Cartucho. Openlabeling: open-source image and video labeler. <https://github.com/Cartucho/OpenLabeling>, 2019.
- [11] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.
- [12] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

- [13] Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. Detect what you can: Detecting and representing objects using holistic models and body parts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1971–1978, 2014.
- [14] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [15] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [16] Gabriela Csurka, Diane Larlus, Florent Perronnin, and France Meylan. What is a good evaluation measure for semantic segmentation?. In *BMVC*, volume 27, page 2013. Citeseer, 2013.
- [17] Stamatia Dasiopoulou, Eirini Giannakidou, Georgios Litos, Polyxeni Malasioti, and Yiannis Kompatsiaris. A survey of semantic image and video annotation tools. In *Knowledge-driven multimedia information extraction and ontology evolution*, pages 196–239. Springer, 2011.
- [18] Datururks. Datururks online tool to build Image Bounding Box, NER, NLP and other ML datasets. <https://datururks.com/index.php>, 2019.
- [19] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.
- [20] Jordi Freixenet, Xavier Muñoz, David Raba, Joan Martí, and Xavier Cufí. Yet another survey on image segmentation: Region and boundary information integration. In *European Conference on Computer Vision*, pages 408–422. Springer, 2002.
- [21] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.
- [22] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.
- [23] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

- [24] Marc Gorriz, Axel Carlier, Emmanuel Faure, and Xavier Giro-i Nieto. Cost-effective active learning for melanoma segmentation. *arXiv preprint arXiv:1711.09168*, 2017.
- [25] Stephen Gould, Richard Fulton, and Daphne Koller. Decomposing a scene into geometric and semantically consistent regions. In *2009 IEEE 12th international conference on computer vision*, pages 1–8. IEEE, 2009.
- [26] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pages 3338–3346, 2014.
- [27] Bharath Hariharan, Pablo Arbelaez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *Proceedings of the 2011 International Conference on Computer Vision, ICCV '11*, pages 991–998, Washington, DC, USA, 2011. IEEE Computer Society.
- [28] Simon S Haykin et al. *Neural networks and learning machines/simon haykin.*, 2009.
- [29] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.
- [30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [31] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.
- [32] BROAD Institute. Broad Bioimage Benchmark Collection. <https://data.broadinstitute.org/bbbc/index.html>, 2019.
- [33] Mark Johnson and Dat Quoc Nguyen. How much data is enough? predicting how accuracy varies with training data size. <http://web.science.mq.edu.au/~mjohnson/papers/Johnson17Power-talk.pdf>, 2017.
- [34] Mamta Juneja and Parvinder Singh Sandhu. Performance evaluation of edge detection techniques for images in spatial domain. *International journal of computer theory and Engineering*, 1(5):614, 2009.
- [35] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

- [36] Pushmeet Kohli, Philip HS Torr, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*, 82(3):302–324, 2009.
- [37] Matej Kristan, Jiri Matas, Aleš Leonardis, Tomas Vojir, Roman Pflugfelder, Gustavo Fernandez, Georg Nebehay, Fatih Porikli, and Luka Čehovin. A novel performance evaluation methodology for single-target trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2137–2155, Nov 2016.
- [38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [39] Inc Labelbox. Labelbox: a collaborative training data platform for artificial intelligence applications. <https://labelbox.com/>, 2019.
- [40] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
- [42] Ce Liu, Jenny Yuen, and Antonio Torralba. Nonparametric scene parsing: Label transfer via dense scene alignment. Institute of Electrical and Electronics Engineers, 2009.
- [43] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.
- [44] Junliang Luo, Sageev Oore, Paul Hollensen, Alan Fine, and Thomas Trappenberg. Self-training for cell segmentation and counting. In *Advances in Artificial Intelligence: 32st Canadian Conference on Artificial Intelligence, Canadian AI 2019, Kingston, ON, Canada, May 28–31, 2019, in press*. Springer, 2019.
- [45] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.
- [46] David R Martin, Charless C Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE transactions on pattern analysis and machine intelligence*, 26(5):530–549, 2004.

- [47] Rottmann Matthias, Kahl Karsten, and Gottschalk Hanno. Deep bayesian active semi-supervised learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 158–164. IEEE, 2018.
- [48] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [49] Anton Milan, Trung Pham, K Vijay, Douglas Morrison, Adam W Tow, L Liu, J Erskine, R Grinover, A Gurman, T Hunn, et al. Semantic segmentation from limited training data. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1908–1915. IEEE, 2018.
- [50] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 1969.
- [51] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.
- [52] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 891–898, 2014.
- [53] George Papandreou, Liang-Chieh Chen, Kevin P Murphy, and Alan L Yuille. Weakly-and semi-supervised learning of a deep convolutional network for semantic image segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1742–1750, 2015.
- [54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [55] Jos BTM Roerdink and Arnold Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta informaticae*, 41(1, 2):187–228, 2000.
- [56] David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017.
- [57] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

- [59] Matthias Rottmann, Karsten Kahl, and Hanno Gottschalk. Deep bayesian active semi-supervised learning. *arXiv preprint arXiv:1803.01216*, 2018.
- [60] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [61] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173, 2008.
- [62] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [63] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [64] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.
- [65] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgb-d images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012.
- [66] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.
- [67] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [68] Carsten Steger, Markus Ulrich, and Christian Wiedemann. *Machine vision algorithms and applications*. John Wiley & Sons, 2018.
- [69] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.
- [70] Vladimír Ulman, Martin Maška, Klas EG Magnusson, Olaf Ronneberger, Carsten Haubold, Nathalie Harder, Pavel Matula, Petr Matula, David Svoboda, Miroslav Radojevic, et al. An objective comparison of cell-tracking algorithms. *Nature methods*, 14(12):1141, 2017.

- [71] Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li. Deep learning for content-based image retrieval: A comprehensive study. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 157–166. ACM, 2014.
- [72] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. *arXiv preprint arXiv:1812.05050*, 2018.
- [73] Paul Werbos. Beyond regression:” new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.
- [74] Westjet. Sample image from. https://www.westjet.com/book/resources/_prod/img/hero/halifax.jpg, 2019.
- [75] W. Xie, J. A. Noble, and A. Zisserman. Microscopy cell counting and detection with fully convolutional regression networks. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 6(3):283–292, 2016.
- [76] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *arXiv preprint arXiv:1608.05442*, 2016.
- [77] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.