BIOMEDICAL IMAGE SEGMENTATION AND OBJECT DETECTION USING DEEP CONVOLUTIONAL NEURAL NETWORKS

by

Liming Wu

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Electrical and Computer Engineering

School of Electrical & Computer Engineering

Hammond, Indiana

May 2019

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Bin Chen, Chair
School of Engineering
Dr. Xiaoli Yang
School of Engineering
Dr. Quamar Niyaz
School of Engineering

Approved by:

Dr. Vijay Devabhaktuni
Head of the department, Electrical and Computer Engineering

ACKNOWLEDGMENTS

The successful completion of this thesis is not my own merit. Although I did most of it, I cannot finish it at the prescribed time without the guidance of my advisor and my friends. Here I want to thanks for their teaching, help, and encouragement to me.

Firstly, I would like to thank my academic advisor Dr. Bin Chen for all his guidance, encouragement and support during my two years' research at Purdue University Northwest. I would also like to thank my dissertation committee Dr. Xiaoli Yang and Dr. Quamar Niyaz for their participation, and timely feedback.

Also, I gratefully thank Dr. Quamar Niyaz for all his constant help, warm-hearted guidance, and valuable suggestions for my future career.

Moreover, I would like to thank Dr. Chenn Zhou and the Center for Innovation through Visualization and Simulation (CIVS) for the first year's funding of being a research assistant. I would like to thank my advisor Dr. Bin Chen for the second year's funding of being a teaching assistant. Also, I would like to thank all the students and faculties who helped me.

Finally, I would like to thank my family especially my dad's financial support and my mom's encouragement. I could not be able to achieve my dreams without their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Author: Wu, Liming. MSECE
Institution: Purdue University
Degree Received: May 2019
Title: Biomedical Image Segmentation and Object Detection Using Deep Convolutional Neural Networks
Committee Chair: Bin Chen

Quick and accurate segmentation and object detection of the biomedical image is the starting point of most disease analysis and understanding of biological processes in medical research. It will enhance drug development and advance medical treatment, especially in cancer-related diseases. However, identifying the objects in the CT or MRI images and labeling them usually takes time even for an experienced person. Currently, there is no automatic detection technique for nucleus identification, pneumonia detection, and fetus brain segmentation. Fortunately, as the successful application of artificial intelligence (AI) in image processing, many challenging tasks are easily solved with deep convolutional neural networks. In light of this, in this thesis, the deep learning based object detection and segmentation methods were implemented to perform the nucleus segmentation, lung segmentation, pneumonia detection, and fetus brain segmentation. The semantic segmentation is achieved by the customized U-Net model, and the instance localization is achieved by Faster R-CNN. The reason we choose U-Net is that such a network can be trained end-to-end, which means the architecture of this network is very simple, straightforward and fast to train. Besides, for this project, the availability of the dataset is limited, which makes U-Net a more suitable choice. We also implemented the Faster R-CNN to achieve the object localization. Finally, we evaluated the performance of the two models and further compared the pros and cons of them. The preliminary results show that deep learning based technique outperforms all existing traditional segmentation algorithms.

Keywords - Deep Learning, Image Segmentation, Object Detection, Biomedical Image Processing.

ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
CNN	Convolutional Neural Network
GPU	Graphic Processing Unit
CPU	Central Processing Unit
HOG	Histogram Oriented Gradient
SVM	Support Vector Machine
FC	Fully Connected Network
MPL	Multi-Layer Perceptron
FCN	Fully Convolutional Network
IoU	Intersection over Union
NMS	Non-maximum Suppression
RoIs	Region of Interests
RPN	Region Proposal Network
BBox	Bounding Box
VGG	Visual Geometry Group
YOLO	You Only Look Once
R-CNN	Regional Convolutional Neural Network
SPPNet	Spatial Pyramid Pooling in Deep Convolutional Networks
ResNet	Deep Residual Learning for Image Recognition
DenseNet	Densely Connected Convolutional Networks
FPN	Feature Pyramid Network
AP	Average Precision
mAP	Mean Average Precision
BP	Back propagation
ANN	Artificial Neural Network
NLLLoss	Negative Log Likelihood Loss
BCELoss	Binary Cross Entropy Loss
SGD	Stochastic Gradient Descent
Adam	Adaptive moment estimation
BET	Brain Extraction Tool

CHAPTER 1. INTRODUCTION

1.1. Structure of the Thesis

This thesis starts with the introduction of the current issue in computer vision and our objectives for solving these problems. In chapter 2 we will make a simple introduction of some widely used algorithm in computer vision. We will discuss the detailed mathematical theories behind them. Then we start talking about deep learning based algorithms such as neural networks and convolutional neural networks. We describe them conceptually and mathematically to make them easy to understand. Finally, we did a literature review of current state-of-the-art deep learning models in computer vision. These cutting-edge models are the foundations of our implementation.

In chapter 3 we move to the experiment setup and implementation part. We begin with the configuration of our workstation, the deep learning frameworks we use, and the installation of the deep learning environment. Then we start discussing the detail of the dataset, which includes the data source and the data preparation. Further, we introduce our customized models based on state-of-the-art models in chapter 2. We discuss how we implement them and the details of the architecture.

In chapter 4, the results of our models are evaluated on different dataset systematically. We provide not only the numerical results but also the analysis for them. We compare the results of our model with the results from other techniques.

In chapter 5, we provide a high-level analysis of our results and the potential improvements to our customized models.

1.2. Challenge in Computer Vision

Computer vision is a simulation of biological vision using computers and related equipment. Its main task is to obtain the three-dimensional information of the corresponding scene by processing the captured pictures or videos, just like humans and many other kinds of creatures do every day. Because computer vision has excellent potential application value in the fields of industrial and agricultural production, geology, astronomy, meteorology, medicine, and military, it has become

more and more critical in the world. At least in the past decade, techniques for solving various problems in the field of computer vision have made significant progress, and some notable problems include image classification, object detection, image segmentation, image generation, and image subtitle generation. In this chapter, I will briefly explain some of these issues and try to compare them from the perspective of how humans interpret images.

1.2.1. Image Classification

Image classification involves only image-based content tag images. There will be a fixed set of input images and the corresponding labels, and the model must predict the label that best fits the images. Although it seems quite simple, it is one of the core issues in the field of computer vision and has a variety of practical applications. In this thesis, we can see that many seemingly different problems in the field of computer vision are the variant of image classification problems. For people, recognizing a visual concept like "cat" is simple, but from the perspective of computer vision algorithms it is just a three-dimensional matrix with the number in the range of (0, 255) indicates the brightness of the pixels.

1.2.2. Object Detection

Object detection in an image involves identifying various sub-images and drawing a bounding box around each identified sub-image. This solution is slightly more complicated than image classification. Nowadays, the deep learning based object detection and recognition has become the mainstream method. The deep neural network model is mainly used as a convolutional neural network CNN. At present, the existing deep learning-based target detection and recognition algorithms can be roughly divided into three categories. First one is the region proposals based detection algorithms, such as R-CNN, Fast-RCNN and Faster-RCNN. The most famous detection method at present is Faster-RCNN, which is a region convolutional neural network. It uses a technique called the region proposal network, which is responsible for fundamentally localizing the regions of the image that need to be classified and processed. This RCNN model was later tuned and more efficient, now called Faster-RCNN. Convolutional neural networks are often used as part of the candidate region approach to generate regions.

1.2.3. Image Segmentation

Image segmentation is a very important branch in computer vision, aim at segmenting the image into different meaningful regions [1]. Currently, there are two major image segmentation categories, semantic segmentation and instance segmentation. Semantic segmentation is the primary task in computer vision. In semantic segmentation, we need to divide visual input into different semantically interpretable categories. "Semantic interpretability" means that classification categories are meaningful in the real world. For example, we might need to distinguish all the pixels in the image that belongs to the car and paint them in blue. However, unsupervised methods like clustering can be used for segmentation, but the results are not necessarily semantic. These methods are not able to subdivide the classes they train but are better at searching for regional boundaries. Semantic segmentation gives us a more detailed understanding of images than image classification or target detection. This understanding is fundamental in many areas such as autonomous driving, robotics, and image search engines. Therefore, the topic discussed in this thesis is the use of deep learning methods for supervised semantic segmentation. Instance segmentation is the task of detecting objects and drawing contours on all the categories in an image. The number of instances is not known in advance, which make the problem more difficult than the semantic segmentation. So far, most of the architectures for instance segmentation are not end-to-end and extremely complex, such as regional-based convolutional neural networks [2]–[5].

CHAPTER 2. BACKGROUND

2.1. Histogram of Oriented Gradient

Before the convolutional neural network came into people's lives, traditional feature-based algorithms played an essential role in object detection. One of the most common-used algorithms, Histogram Oriented Gradient (HOG), is an image descriptor for solving human target detection that is used to characterize the local gradient direction of the image and represents the features of gradient density distribution[6]. The main idea is that without knowing the specific location of the pedestrian on the image, the distribution of the edge features is also good enough to represent the contour of the pedestrian. The HOG feature extraction process can be divided into several steps.

The first part of the HOG algorithm is color space normalization, which includes image graying and gamma correction. For RGB channel images, HOG needs to convert them to gray images using the formula

$$gray = 0.3 * R + 0.59 * G + 0.11 * B \tag{2.1}$$

To improve the robustness of the detector to interference factors such as illumination, it is necessary to perform gamma correction on the image to complete the normalization of the entire image. This aims at adjusting the contrast of the entire image, reducing the influence of local illumination and shadow, and reducing the overall noise as well. The gamma correction is achieved by the following formula:

$$Y(x, y) = I(x, y)^{\gamma} \tag{2.2}$$

where usually choose $\gamma = 0.5$, and the pixel values are normalized from range (0, 255) to range (0, 15.97).

Pixel gradient calculation. After the color space normalization, calculate the gradient horizontally and vertically. The horizontal operator and vertical operator are $[-1, 0, 1]$ and $[-1, 0, 1]^T$, respectively. The horizontal and vertical gradients of the image at the pixel (x, y) are:

$$\begin{cases} G_x(x,y) = G(x+1,y) - G(x-1,y) \\ G_y(x,y) = G(x,y+1) - G(x,y-1) \end{cases} \tag{2.3}$$

where $G_x(x,y)$ and $G_y(x,y)$ represent the gradient value of current pixel (x, y) in the horizontal direction and vertical direction. Next, calculate the gradient magnitude and gradient direction at the pixel (x, y),

$$\begin{cases} \nabla G(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \\ \theta(x,y) = tan^{-1}\dfrac{G_y(x,y)}{G_x(x,y)} \end{cases} \tag{2.4}$$

Calculate the cell unit gradient direction histogram. Divide the image into several cells, and make sure the adjacent cells do not overlap. The gradient direction is mapped to a range of 180 degrees. The gradient magnitude of the pixel is projected as a weight, and the gradient direction is used to determine which dimension to project. If the gradient direction of the pixel is 20 degrees and the gradient amplitude is 10, then the second dimension of the histogram is increased by 10.

Count the gradient direction histogram for the block. The gradient histogram in each cell unit is counted to form a descriptor for each cell unit. A larger descriptor is composed of cells, called a block. The feature vectors of four cells in a block are connected in series to form the block. According to a cell unit, the gradient direction histogram is a 9-dimensional Hog feature. The Hog characteristic of a block is 4x9=36 dimensions. Due to changes in local illumination and foreground-background contrast, the gradient intensity varies greatly, which requires local contrast normalization of the gradient. The strategy here is to normalize the contrast for each block, typically using L2-norm. Calculate the gradient direction histogram for the window by concatenating the HOG feature vectors of all the blocks in the window. This is then followed by calculating the gradient direction histogram for the whole image, which is composed of many windows. Then, the HOG features of the image are obtained. As long as the feature of an image can be represented, a classification problem can be performed using Support Vector Machine (SVM).

2.2. Classification

2.2.1. Logistic Regression

As one of the most straightforward classification algorithm, logistic regression has been widely used in statistics and machine learning [7]. Logistic Regression is the combination of linear regression and sigmoid function [8]. Before talking about the mathematical formula of logistic regression, let us take a look at the formula for linear regression:

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \boldsymbol{\theta}^T \boldsymbol{x} \tag{2.5}$$

where $\boldsymbol{\theta}$ is the learned parameters and $\boldsymbol{x}$ are the feature vectors. The difference between logistic regression and linear regression is that the prediction value from the logistic model is non-linear because of the characteristic of the sigmoid function, which maps the value range from -∞ to +∞ to range from 0 to 1. The formula for logistic regression is:

$$\begin{aligned} h_\theta(\boldsymbol{x}) &= g(\boldsymbol{\theta}^T \boldsymbol{x}) \\ g(z) &= \frac{1}{1 + e^{-z}} \end{aligned} \tag{2.6}$$

Because of the output of $h_\theta(\boldsymbol{x})$ is between 0 and 1, it is easy to see that the decision boundary is the hyperplane where $h_\theta(\boldsymbol{x}) = 0.5$. Next, we need to calculate the value for parameter θ by Maximum Likelihood Estimation (MLE). According to the probability theory, the probability function is:

$$P(y|x;\theta) = (h_\theta(x))^y * (1 - h_\theta(x))^{1-y} \tag{2.7}$$

Since the m sample data are independent, their joint distribution can be expressed as the product of the marginal distributions. The likelihood function is:

$$L(\theta) = \prod_{i=1}^{m} P\left(y^{(i)} \middle| x^{(i)}; \theta\right) \tag{2.8}$$

$$L(\theta) = \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} * (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

Apply logarithm on both sides:

$$l(\theta) = log(L(\theta)) = \sum_{i=1}^{m} y^{(i)} log\left(h_\theta(x^{(i)})\right) + (1 - y^{(i)}) log\left(1 - h_\theta(x^{(i)})\right) \quad (2.9)$$

So, the cost function would be:

$$J(\theta) = -\frac{1}{m} l(\theta) \quad (2.10)$$

Then, use the gradient descent to find the minimum value of the cost function.

2.2.2. Support Vector Machine

Another popular classification algorithm is the support vector machine (SVM) [9], which is a very powerful and supervised learning algorithm for classification and regression [10]. Although the SVM was invented in the nineteenth century, it is still popular in data analysis and machine learning areas today. Compared to other regression algorithms, such as logistic regression and neural network, the SVM provides a cleaner, and sometimes more powerful, way for learning complex non-linear functions. Sometimes people define SVM as large margin classifiers [11] because its learning strategy is trying to find the largest margin between different samples to optimize the classification problem.

2.3. Neural Networks

The terminology, *Neural Network,* was first proposed in biology fields where people were working on simulating how neural networks work in human brains [12], [13]. The first time that the artificial neural network gained popularity was in the early 1960s when a computer scientist named Rosenblatt proposed the Perceptron, which is a single layer neural network for binary classification problems [14]. The Perceptron might be the first artificial neural network in the world. Many people think that by using thousands of neurons, any problem can be solved. In 1969, Minsky and

Papert, two of the founders of artificial intelligence, published a book called Perceptron [15]. The book pointed out that simple neural networks can only be applied to solving linear problems, and networks that can solve nonlinear problems should have a hidden layer. However, it is theoretically impossible to prove that it is meaningful to extend the perceptron model to a multi-layer network. The neural network popularity lasted for ten years before the book, Perceptron, was published. Since then, researchers in the field of neural networks have been significantly reduced, but there are still a handful of scholars who are still committed to the study of neural networks during difficult times.

2.3.1. Perceptions

From a neural network perspective, the perceptron can also be viewed as a single neuron. However, the single layer perceptron cannot solve the non-linear problem. In the 1980s, the Japanese scientist Kunihiko Fukushima proposed the conception *Neocognitron*, which aims at constructing a pattern recognition model that can simulate the mechanism of the human brain [16]. Many of the innovative ideas about the human visual system and artificial neural network in his work can be considered as the prototype of the convolutional neural network. Figure 2-1 shows the architecture of a multi-layer neural network.

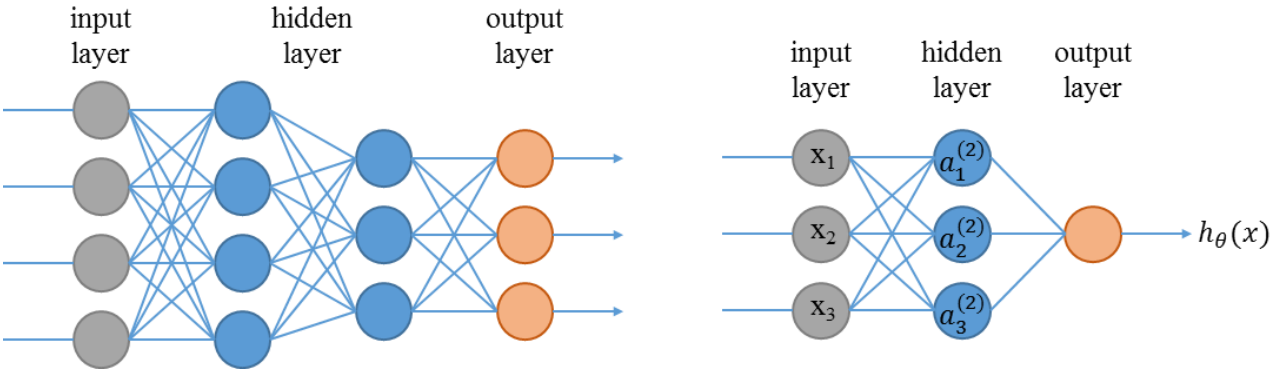

Figure 2-1. A fully connected neural network with two hidden layers (left) and a fully connected neural network with one hidden layer (right).

The multi-Layer Perceptron (MPL) can be regarded as a multi-layer neural network, consists of one input layer, one output layer and multi hidden layers [17], [18]. The MPL can map an input vector X to an output vector Y by learning a non-linear function that overcomes the shortage of single layer perceptron. The neural network works on two stages, one is forward propagation, and the other is backward propagation. For each neuron, there are two parameters called weight and

bias, and a function called activation function. For example, in Figure 2-1, define $a_i^{(j)}$ as the "activation" of unit i in layer j, and $\theta^{(j)}$ as the matrix of weights controlling function mapping from layer j to layer j+1. Then the values for the activation nodes are obtained as follows:

$$\begin{aligned} a_1^{(2)} &= g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) \\ a_2^{(2)} &= g\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right) \\ a_3^{(2)} &= g\left(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3\right) \\ h_\theta(x) &= a_1^{(3)} = \left(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right) \end{aligned} \tag{2.11}$$

Note that if the network has s_j units in layer j, s_{j+1} units in layer j+1, then $\theta^{(j)}$ will be of dimension $s_{j+1} * (s_j + 1)$.

2.3.2. Activation function

The activation function plays an essential role in the information transmission of the neural network. The output value for any neurons can be anything range from –inf to +inf. Therefore, whether a neuron should be activated cannot be determined because the neuron does not know the bound of the value. The activation function can constraint the output value to a fixed range [19]. The following are some widely used activation functions.

The sigmoid function is the first activation function used in the perceptron and still widely used in deep neural networks. [20]. The problem for the sigmoid function is that the gradients on the two sides are going to be very small which makes the network stop learning. For deep neural networks, the gradients may disappear which caused information loss.

$$A = \frac{1}{1 + e^{-x}} \tag{2.12}$$

Tanh function.

$$f(x) = tanh(x) = \frac{2}{1 + e^{-x}} - 1 \tag{2.13}$$

The Tanh function is just a scaled formation of a sigmoid function, so it has the same problem as the sigmoid function.

$$tanh(x) = 2sigmoid(2x) - 1 \tag{2.14}$$

ReLU function simulates how brain neuron works. Unilateral inhibition, relatively broad excitatory boundary and sparse activation. Relu will make part of neurons output zero values, which will cause the sparseness of the network, and reduce the interdependence of parameters, alleviating the occurrence of over-fitting problems [21]. Usually, ReLU is used in hidden layers.

$$A(x) = max(0, x) \tag{2.15}$$

Leaky ReLU is similar as ReLU but avoids zero gradient when input is less than zero.

$$f(x) = \begin{cases} x, & if\ x > 0 \\ ax, & else \end{cases} \tag{2.16}$$

2.3.3. Loss function

In Machine learning, the loss function, sometimes called cost function, is a function that measures the errors between the prediction and ground truth. The loss function is a non-negative mathematical function that used to measure the inconsistency between the predicted value f(X) of the model and the ground truth value y, usually expressed by L(y, f(X)). The smaller the loss function, the more robust the model is. The loss function is the core part of the empirical risk function and an important part of the structural risk function [18]. The fundamental risk function of the model includes empirical risk terms and regular terms, which can usually be expressed as follows:

$$\theta^* = arg\,\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i;\theta) + \lambda\Phi(\theta)) \tag{2.17}$$

Where the mean function represents the empirical risk function, L represents the loss function, and Φ is the regularization term. The meaning of the overall equation is to find the proper θ such that the equation has the smallest value. In this thesis, I mainly used the softmax function, the negative log-likelihood loss function, the cross-entropy loss function, and the binary cross-entropy loss function.

Softmax function that can convert an N-dimensional vector to another dimension k and make every output tensor lie in the range of (0, 1) and the sum of all elements are 1 [22].

$$Softmax(x_i) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; for\; j = 1, \dots, K. \tag{2.18}$$

where K is number of classes.

Log softmax is the similar as softmax except wrapped by a Log function.

$$LogSoftmax(x_i) = log\left(\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}\right) for\; j = 1, \dots, K. \tag{2.19}$$

The negative log-likelihood loss function (NLLLoss) is widely used in classification problems. Compare with cross-entropy loss, the NLLLoss does not require one hot encoding on the ground truth label but need a log-probabilities input, which is easily obtained by adding an additional log softmax layer before NLLLoss calculation. The loss function can be represented as:

$$\begin{gathered} L = \ell(x, y) = \sum_{n=1}^{N} l_n, \\ l_n = -x_{n,y_n}, \end{gathered} \tag{2.20}$$

The cross-entropy comes from Shannon's information theory. In simple terms, cross entropy is used to measure the uncertainty of the system using the strategy f(x) specified by the non-real

distribution q_k under a given real distribution p_k [23]. The lower the cross entropy, the better the strategy is. We always minimize the cross-entropy because if the cross-entropy is lower, the generated distribution by the algorithm is closer to the real distribution. Cross-entropy loss function combines the log softmax function and NLLLoss function. It is useful for training a classifier. When using this loss function, there is no need to add a softmax layer, but it requires one-hot encoding which means only one position has the value 1 and the rest are 0.

$$loss(x, class) = -\log\left(\frac{e^{(x[class])}}{\sum_j e^{(x[j])}}\right) = -x[class] + \log\left(\sum_j e^{(x[j])}\right) \tag{2.21}$$

The Binary Cross Entropy Loss (BCELoss) is usually used to measure the loss of a reconstruction for each pixel, widely used in image segmentation, image reconstruction such as auto-encoder [24].

$$\begin{gathered} L = \ell(x, y) = \sum_{n=1}^{N} l_n, \\ l_n = -w_n[y_n * \log(x_n) + (1 - y_n) * \log(1 - x_n)], \end{gathered} \tag{2.22}$$

where N is the batch size and w is the vector that represents the weight for each class. Note: the value of target y should be a within range (0, 1). So, before the BCELoss function, a sigmoid function is usually used to convert the predicted value to the range of (0, 1).

2.3.4. Back Propagation Algorithm

The backpropagation algorithm is an innovative supervised algorithm invented by Rumelhart, Hinton, and Williams in 1986 [25], [26]. It has become a standard method to train the neural network by backpropagating the gradient to update the weights and bias for each neuron. A trained neural network can output correct results by taking the input data. However, due to the complexity of the non-linear function that the neural network simulates, it is complicated to calculate the value of weights and bias for each neuron mathematically. The backpropagation algorithm, a conceptually easy method, provides a solution for iteratively determining the parameters of the neuron. The original backpropagation algorithm employees the gradient descent optimization, which is a time-consuming but effective method, to update the parameters. The gradient descent method is a straightforward algorithm that works very well in common practice although it does

not guarantee to find the minimum error location. The gradient descent optimization has become the most widely used optimization algorithm in the neural network. For minimizing the error, only need to advance the parameter by a step in the opposite direction of the gradient to achieve the descent of the loss function.

$$\nabla_\theta J(\theta) = \frac{dJ(\theta)}{d\theta} = \lim_{h \to 0} \frac{J(\theta + h) - f(\theta)}{h} \tag{2.23}$$

where $\nabla_\theta J(\theta)$ is the gradient of the parameters.

According to the different amount of data used in the calculation of the objective function, the gradient descent algorithm can be divided into batch gradient descent,

$$\theta \leftarrow \theta - \eta * \nabla_\theta J(\theta) \tag{2.24}$$

stochastic gradient descent (SGD),

$$\theta \leftarrow \theta - \eta * \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right) \tag{2.25}$$

and mini-batch gradient descent.

$$\theta \leftarrow \theta - \eta * \nabla_\theta J\left(\theta; x^{(i:i+n)}; y^{(i:i+n)}\right) \tag{2.26}$$

where n is the mini-batch size.

For batch gradient descent, the loss is calculated by taking the whole dataset as input and then update the weights. The SGD only take one sample from the entire dataset and perform an update, which makes the weights of the network update very frequently and causes the objective function to fluctuate heavily [27]. The mini-batch, a combination of batch gradient descent and stochastic (SGD), takes a small batch of training examples and performs an update. This mechanism can make use of parallel computing in the GPU kernel [28]. Usually, the mini-batch size ranges from 16 to 256. The following is the pseudo code for gradient descent algorithm in a one hidden layer network.

Table 2-1. Backpropagation algorithm [29].

Table 2-1 The backpropagation algorithm for a three layer network

Initialize the network weights (often small random normal distribution values)
do
forEach training example named input and target
    prediction = forward_pass(network_model, input)
    compute error loss(prediction, target) at the output units
    compute Δw_h for all weights from hidden layer to output layer
    compute Δw_i for all weights from input layer to hidden layer
    update network weights in each layer
endFor until all data classified correctly or another stopping criterion satisfied
Return the network

2.3.5. Optimization Algorithms

For the neural network model, the gradient algorithm can be efficiently calculated using the BP algorithm, thereby achieves gradient descent optimization. However, one problem of gradient descent is that it does not guarantee to find the minimum error location, usually stuck in a local minimum or saddle point. Thus, the hyperparameter tuning becomes a vital step. Fortunately, there are many variants of the classic gradient descent algorithm that deal with this problem such as learning rate decay, weight decay, and momentum.

2.3.5.1. Momentum optimization

The momentum optimization algorithm was proposed by Boris Polyak in 1964. It is based on the physical fact that a small ball is rolled down from the top of the mountain with a very small initial velocity, then the velocity increases rapidly under the action of gravity acceleration, and finally reach a stable speed due to the resistance. The update equation for the momentum optimization is

$$\begin{aligned} \boldsymbol{m} &\leftarrow \gamma * \boldsymbol{m} + \eta * \nabla_\theta J(\theta) \\ \theta &\leftarrow \theta - \boldsymbol{m} \end{aligned} \tag{2.27}$$

where the hyperparameter γ is the momentum term, usually set to around 0.9. As the formula shows, the momentum term will increase if the gradient has the same direction with the momentum, otherwise decrease, which helps the convergence of the network [30].

2.3.5.2. RMSprop

The RMSprop was proposed by Hinton in his lecture, aims at solving too large learning rate. It introduced another hyperparameter to divide the learning rate by an exponentially decaying average of squared gradients,

$$\begin{aligned} \boldsymbol{s} &\leftarrow \gamma * \boldsymbol{s} + (1-\gamma) * \nabla_\theta J(\theta) \odot \nabla_\theta J(\theta) \\ \theta &\leftarrow \theta - \frac{\eta}{\sqrt{\boldsymbol{s}+\epsilon}} \odot \nabla_\theta J(\theta) \end{aligned} \tag{2.28}$$

where $\odot$ represents the element wise multiplication.

2.3.5.3. Adaptive moment estimation (Adam)

The Adam algorithm, a combination of momentum and RMSprop, was proposed by Kingma in 2015 [31], shows excellent performance in practice while training deep neural networks. It introduced two hyperparameters β_1 and β_2 to adjust the exponentially decaying average of past gradients m_t .

$$\begin{aligned} \boldsymbol{m} &\leftarrow \beta_1 * \boldsymbol{m} + (1-\beta_1) * \nabla_\theta J(\theta) \\ \boldsymbol{s} &\leftarrow \beta_2 * \boldsymbol{s} + (1-\beta_2) * \nabla_\theta J(\theta) \odot \nabla_\theta J(\theta) \\ \boldsymbol{m} &\leftarrow \frac{\boldsymbol{m}}{1-\beta_1} \\ \boldsymbol{s} &\leftarrow \frac{\boldsymbol{s}}{1-\beta_2} \\ \theta &\leftarrow \theta - \frac{\eta}{\sqrt{s+\epsilon}} \odot \boldsymbol{m} \end{aligned} \tag{2.29}$$

Usually, the hyper parameters β_1, β_2 and ϵ are set to 0.9, 0.999 and 1e-8 respectively.

2.4. Convolutional Neural Networks

Convolutional neural networks (CNN) are an artificial neural network structure that has gradually emerged in recent years. Because convolutional neural networks can give better prediction results in image and speech recognition, this technology is also widely used [32]–[34]. The most commonly used aspect of convolutional neural networks is computer image recognition and classification [35]. Because of constant innovation, it is also used in video analysis, natural language processing, drug discovery and so on. The most recent Alpha Go, let the computer see Knowing Go, there is also the use of this technology. Image classification is a task of classifying an input image (cat, dog...) into a category that best describes an image feature. For humans, the recognition of the images is our natural skill and we can quickly identify the environment and objects around us without hesitation. However, when input an image to a computer, what it sees is a three-dimensional matrix that represents the pixels values of an image. Although these numbers do not make sense for us to classify images, they are the only data that the computer obtains when the image is input. Concretely, the CNN consists of four different building blocks, which are convolutional layers, max pooling layer, flatten and fully connected layers.

2.4.1. Problem of Fully Connected Networks

The convolutional layer is the core component of CNN architecture. Compare with the convolutional neural network; the fully connected network is not good at image recognition tasks. There are too many parameters in the fully connected network. A 1024*1024*3 dimensional image has ~3 million pixels, which makes 3 million nodes in the input layer. If the first hidden layer has 100 nodes, then this hidden layer will end up having 300 million parameters which are a huge memory consumption and poor scalability.

Moreover, the FC does not utilize the spatial location information between different pixels. For image recognition tasks, the connection between each pixel and its surrounding pixels is relatively tight, and the connection to pixels far away may be small. If a neuron is connected to all neurons in the previous layer, it is equivalent to treating all the pixels of the image equally for one pixel, which is not in line with the earlier assumptions.

2.4.2. Convolutional Neural Networks

Trying to learn many weights that are not important, such learning will be very inefficient. Three methods are used to avoid the disadvantages described above. The most intuitive one is the local connection. In image recognition, some patterns are much smaller than the whole image. When detecting an object in the image, instead of look on the entire image, only a small region needs to be checked to see if it has the feature of the object. Thus, in CNN, each neuron is no longer connected to all neurons in the previous layer, but only to a small number of neurons, which reduces many parameters. Another one to reduce the parameters is weight sharing, which is a set of connections share the same weights rather than having different weights for each connection, which significantly reduced the number of parameters. Downsampling is also an intuitive way to simplify the network. When an image is downsampled by removing the odd columns and even rows, it still can be recognized by human, which means subsampling does not have any influence for image recognition. Thus, subsampling can be used to reduce the number of samples per layer, further reducing the number of parameters, and improving the robustness of the model as well. Sometimes, paddings are used to avoid the shrink of the image size during the convolution phase. For each convolution layer, three parameters, kernel size, kernel stride, and padding, are specified.

The pooling layer, usually following by the convolution layer, is used to downsample the features spatially. During the process of convolution, the neural network may lose some information. However, the pooling layer can solve this problem by filter useful information and analyze them in the next convolution layer, and reduces the computational burden by reducing the number of parameters as well.

The convolution layers and pooling layers may repeat many times to extract more advanced features. Finally, the three-dimensional feature is flattened to a one-dimensional feature and send to a fully connected network. The fully connected network's job is to classify these features to a specific category.

2.4.3. Convolution Arithmetic Formula

When defining a convolution layer, some parameters should be initialized such as kernel size, kernel stride, and padding size. The output size of the feature map will be affected by these

parameters and the input size of the image as well. If the input image size, kernel size, kernel stride, and padding size are defined by i, k, s, p. Then

$$o = \frac{i + 2p - k}{s} + 1 \tag{2.30}$$

where o is the output size of the feature map, here assuming the width and height of the input image are the same.

2.4.4. Transposed Convolution

The conception of transposed convolution was firstly proposed by Zeiller [36] in 2010, which specified it as deconvolution. With the successful application of deconvolution in neural network visualization, it has been adopted by more and more work such as scene segmentation, generative model and so on. Among them, deconvolution has many other names, such as transposed convolution, fractional stride convolution, and so on. As discussed above, the convolution layer is used to extract the features; the transposed convolution is a reversed process of convolution, used to up-sample the feature. If regarding the convolution layer as an encoder that encodes the image to a feature map, the transposed convolution layer is a decoder that reconstructs the original image from the feature map. Thus, the transposed convolution is usually used in image reconstruction, image up-sampling or image generation.

The conceptual way to understand transposed convolution is thinking about given a 2D image, e.g., 8x8 with kernel size 2x2, stride 2 and padding 0. Each pixel in the 8x8 image is enlarged to 2x2 by element-wise multiplication with the 2x2 kernel. So the 8x8 image is enlarged to 16x16. Another way to calculate the output shape of transposed convolution is to calculate o' in formula (2.31. Also assume i' is the input dimension. So the output dimension is:

$$o' = (i' - 1)s + k - 2p \tag{2.31}$$

Let us consider a transposed convolution with 3x3 kernel over a 5x5 input feature. E.g. ($i' = 5, k = 3, s = 1, p = 0$). Each pixel on the input feature map will be diluted to a 3x3 feature by multiplying with the 3x3 kernel. Then, the same calculation applies to the second pixel on the input feature. Since the stride is set to 1, the 3x3 feature generated by the second pixel will shift one position. So

2/3 of the 3x3 feature generated by the first pixel is overridden by the 3x3 feature generated by the second pixel.

2.5. The state-of-the-art models

In this section, I will introduce some state-of-the-art models that we used in the experiment of this thesis that achieves excellent results in image recognitison challenges.

2.5.1. VGG

The first well-known model VGGNet was proposed by Visual Geometry Group of Oxford University [37]. They proposed a series of VGG models in different complexity (from VGG16 – VGG19), which achieves excellent results in face recognition and image classification on ImageNet challenge. The original intention of VGG to study the depth of the convolutional network is to find out how the depth of the convolutional network affects the accuracy and accuracy of large-scale image classification and recognition, which is also the main contribution of VGG. Compare with the Alexnet model [35], VGG has done more in-depth research on the depth and width of the deep neural network. It is generally believed that deeper networks have stronger expressive power than shallow networks, and can accomplish more complex tasks. However, one problem of the deep network is the huge number of parameters. To avoid this problem, VGG uses 3x3 small convolution kernels in all layers with kernel stride 1 and padding 1. According to the author's analysis, a 3x3 convolution kernel is sufficient to capture the horizontal, vertical, and diagonal changes of the pixel values. The use of large convolution kernels will cause an explosion of parameter quantities, and some parts of the image will be convolved multiple times, which may cause difficulty in feature extraction. The dimension of the input image is set to a 224x224 RGB image. VGG has three fully connected layers, and the different number of convolution layers depends on the version of the model (VGG11 - VGG19). The simplest model VGG11 has eight convolutional layers and three fully connected layers, and the most complicated VGG19 has 16 convolutional layers and three fully connected layers. Also, the convolution layers in VGG does not necessarily follow by the pooling layer. The detailed architecture of VGG is shown in Table 2-2. VGG architecture configuration, where conv represents convolution layers, FC stands for fully connected layers, conv3-64 means convolution layers with 3x3 filters and have depth 64. All convolutional layers are followed by a nonlinear activation function layer [37].

Although VGG does not have the highest accuracy in the image recognition challenges, it is an excellent feature extractor. We use the VGG16 as the backbone of our model in the experiment of the thesis.

Table 2-2. VGG architecture configuration, where conv represents convolution layers, FC stands for fully connected layers, conv3-64 means convolution layers with 3x3 filters and have depth 64. All convolutional layers are followed by a nonlinear activation function layer [37].

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
Input (224 x 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
Maxpooling					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
Maxpooling					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
Maxpooling					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
Maxpooling					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
Maxpooling					
FC-4096					
FC-4096					
FC-1000					
Softmax					

2.5.2. Residual Networks

As the increase in the number of neural network layers, the accuracy of the training set decreases. This is not caused by the over-fitting problem because an over-fitted network should achieve very high accuracy on the training set and low accuracy on the testing set. Thus, an innovative model called Deep Residual Network was proposed by Kaiming in 2015 [38]. The most fundamental motivation for ResNet is the so-called "degradation" problem, in which the error rate increases as the model level deepens. However, the depth of the model is deepened and the learning ability is enhanced, so a deeper model should not produce a higher error rate than a shallower model. The reason for this "degradation" problem is attributed to the optimization problem. When the model becomes complex, the optimization of SGD becomes more difficult, resulting in a lack of ability to find the optimal results.

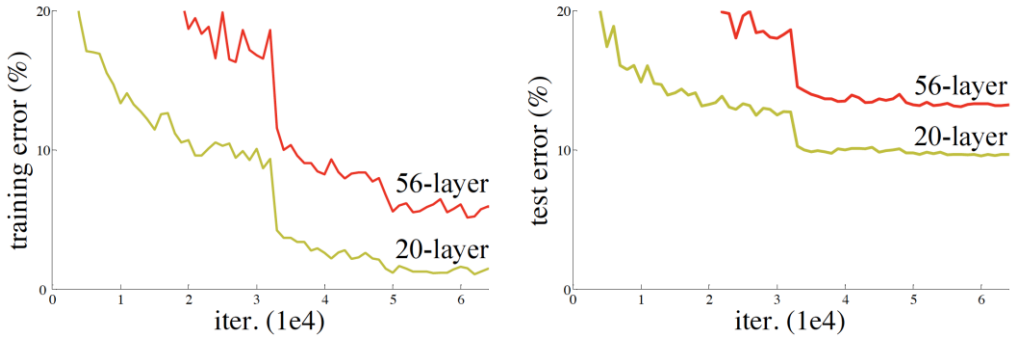

Figure 2-2. The error on the training set (left) and the testing set (right) for CIFAR-10 dataset with 20-layer and 56-layer “plain” network. When the number of layers for the network increases from 20 to 56, the results become worse [39].

The basic architecture of ResNet is the identity mapping, that convert the original mapping from $H(x)$ to $F(x) + x$.

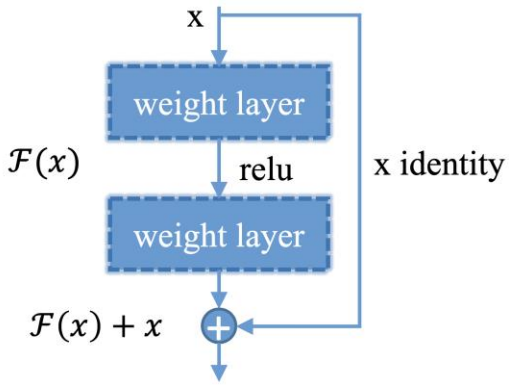

Figure 2-3. The building block of the residual learning.

The author thinks that this will ease the optimization tasks but not affecting the training results. The authors assume that the optimization of F(x) will be much simpler than H(x). This idea is also derived from the residual vector coding in image processing. Through a reformulation, a problem is decomposed into multiple scales direct residual problems, which can well optimize the training effect. This Residual block is achieved by a shortcut connection. The output of this block is added element-wisely by shortcut. This simple addition does not add extra parameters and calculations to the network, but can significantly increase the model's training speed and further improve training effect. So, when the number of layers of the model increased, this simple structure can solve the degradation problem well [40]. The author proves his idea by comparing the training results on an 18 layer and a 34 layer network without shortcuts and with shortcuts. It turns out that the network with shortcuts achieves better results and has a smaller amount of parameter compare with VGG Network. In this experiment, except for the VGG Network, the ResNet101 as the backbone of the Faster R-CNN network was implemented.

The deep residual network consists of many identical building blocks. The author did the experiments on the different depth of the network, found that as the depth goes deeper, the network achieves better results. The following table shows the detailed architecture of different depth residual network.

Table 2-3. The architecture of the deep residual network for different depth. FLOPs are the number of floating point operations per second [38].

Layer name	Output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112x112	7x7, 64, stride 2				
		3x3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3,64 \\ 3 \times 3,64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3,64 \\ 3 \times 3,64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,64 \\ 3 \times 3,64 \\ 1 \times 1,256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,64 \\ 3 \times 3,64 \\ 1 \times 1,256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,64 \\ 3 \times 3,64 \\ 1 \times 1,256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3,128 \\ 3 \times 3,128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3,128 \\ 3 \times 3,128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1,128 \\ 3 \times 3,128 \\ 1 \times 1,512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1,128 \\ 3 \times 3,128 \\ 1 \times 1,512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1,128 \\ 3 \times 3,128 \\ 1 \times 1,512 \end{bmatrix} \times 4$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3,256 \\ 3 \times 3,256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3,256 \\ 3 \times 3,256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1,256 \\ 3 \times 3,256 \\ 1 \times 1,1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1,256 \\ 3 \times 3,256 \\ 1 \times 1,1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1,256 \\ 3 \times 3,256 \\ 1 \times 1,1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3,512 \\ 3 \times 3,512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3,512 \\ 3 \times 3,512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,512 \\ 3 \times 3,512 \\ 1 \times 1,2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,512 \\ 3 \times 3,512 \\ 1 \times 1,2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1,512 \\ 3 \times 3,512 \\ 1 \times 1,2048 \end{bmatrix} \times 3$
	1×1					
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

2.5.3. Feature Pyramid Networks (FPN)

For most of the object detection models such as Faster R-CNN, YOLO, detecting the different scale of objects is still a challenging task [3], [41]. The idea of FPN is straightforward, which is generating the features in different scales and detecting the objects on these features separately [42]. FPN mainly solves the multi-scale problem in object detection. Through simple network connection changes, the performance of small object detection is greatly improved without substantially increasing the calculation amount of the original model.

In the object detection, in the case of limited calculation, the depth of the network (corresponding to the receptive field) and stride are usually a contradiction. The down-sampling rate of the most common network structure is generally larger (such as 32). Small objects can even be smaller than the size of stride, which resulting in a sharp drop in the detection performance of small objects. Previously, there are two methods to solve this problem. Almost all methods that have achieved excellent results on ImageNet and COCO inspection tasks use the Image Pyramid approach. However, such a method is difficult to apply in practice due to high time and computational consumption. The SSD detection framework uses a similar idea [43]. The problem with such a method is to directly force the different layers to learn the same semantic information. For convolutional neural networks, different depths correspond to different levels of semantic features; shallow network resolution is high, more detailed features are learned, and deep network resolution is low, so more semantic features are learned.

The overall architecture of FPN is similar to U-Net model except FPN replaced the feature concatenation with feature element-wise addition, and use ResNet as the feature extractor. The FPN uses ResNet in its down-sampling part to extract features. This consists of many convolution blocks, and the spatial dimensions of the features are reduced by $\frac{1}{2}$ after each block. The output of each block is labeled as C_i ($C_2 - C_5$), as is shown in Figure 2-4. The features $C_2 - C_5$ are used as the pyramid features to feed into the FPN. The 1x1 convolution kernel is applied to the feature C_5 to generate feature M_5. Then a 3x3 convolution kernel is applied on M_5 to generate P_5, which becomes the first scale of feature map for object detection. Then the feature M_5 is up-sampled, and an element-wise addition is applied on the up-sampled M_5 and the 1x1 convolved C_4. As we keep repeating the previous steps, the different scale of features ($P_2 - P_5$) is generated for object

detection. The whole process is stopped on P_2 is because there will be a large amount of computation and memory consumption if calculating P_1.

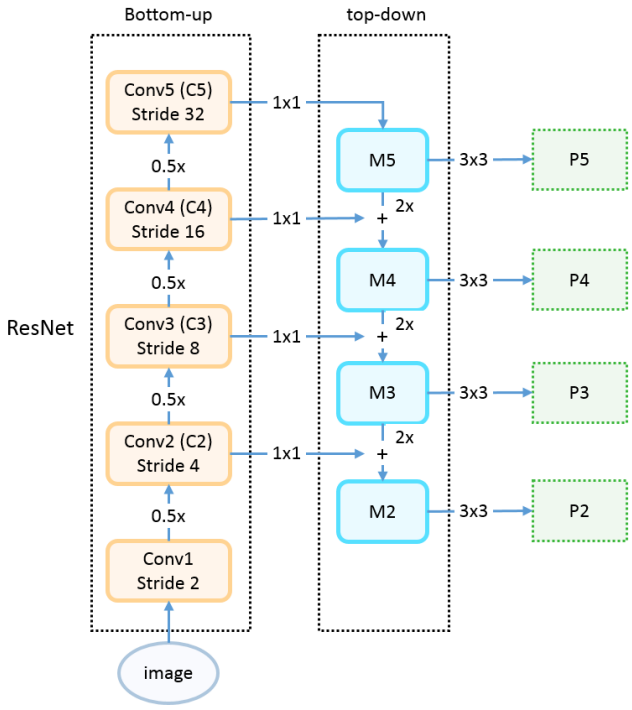

Figure 2-4. The architecture of feature pyramid network [44].

In this thesis, we tested FPN on fetus brain segmentation. The ResNet is used as the feature extractor backbone to generate $C_2, C_3, C_4\ and\ C_5$. The feature $P_2, P_3, P_4\ and\ P_5$ are up-sampled to have the same dimension as the input image, and the up-sampled features are concatenated as one feature. Then this feature is convolved to have exactly the same dimension as the input image. Finally, the output feature is passed into a sigmoid layer to map the value to a probability (0-1) and a binary cross-entropy is implemented to calculate the loss between the output probability and the target mask.

2.5.4. Fully Convolutional Networks

Semantic segmentation is an important part of image processing and image understanding in computer vision, and also an essential branch in the field of AI. Semantic segmentation is to classify each pixel in the image, determine the category of each point (such as belonging to the background, people cars and so on.), and then divide the region. At present, semantic segmentation has been widely used in scenarios such as automatic driving and drone determination. Different

from the classification problem, image semantic segmentation needs to determine the category of each pixel of the image and perform precise segmentation. However, during the convolution and pooling, the CNN will lose the details of the image because of the shrink of the feature map, which makes it difficult to represent the contour of the object.

The Fully Convolutional Network (FCN) was proposed by Jonathan Long to ease the research work in semantic segmentation [45]. Since then, FCN has become the basic framework of semantic segmentation, and subsequent algorithms are derived from this framework. Usually, there are several fully connected layers followed by the convolution layers to map the feature map to a fixed length feature vector. The traditional CNN structure represented by VGG is suitable for image-level classification and regression tasks because it can output a probability vector of the entire input image. For example, a VGG model for ImageNet outputs a 1000-dimensional vector to represent the probability that the input image belongs to each class. Unlike the CNN, the FCN can accept the input images in any size. The feature maps are up-sampled by the transposed convolution layer to restore it to the same size of the input image so that a prediction can be generated for each pixel while preserving the spatial information in the original input image. Finally, a pixel-level classification is performed on the up-sampled feature maps. The difference between FCN and CNN is that the last fully connected layer in CNN is replaced by the convolutional layers, and the output is a picture with a good Label, and this picture can be semantically segmented.

2.5.5. U-Net

U-Net is a segmentation network proposed by Olaf Ronneberger in the ISBI challenge [46]. The architecture of U-Net and FCN are very similar except U-Net employees the concatenation of the feature map in convolution layer and transpose convolution layer. Also, the U-Net can be trained well in a small dataset (about 30 pictures).

The entire U-Net network structure is similar to an uppercased U letter. First convolution and pooling down-sampling followed by the transposed convolution for up-sampling, then the lower layer feature map is concatenated with the higher layer feature map, then up-sampling again. Repeat this process until getting the feature map of output 388x388x2, and finally get the output segment map via softmax function. It is very similar to the FCN idea from the overall architecture.

In this thesis, the segmentation tasks on the nucleus and lung dataset are mainly based on the U-Net model. The biomedical dataset is usually smaller compared with the ImageNet dataset.

2.5.6. Deep Convolutional Generative Adversarial Networks

The generative adversarial network (GAN) was proposed by Ian J. Goodfellow in 2014 that gains much popularity in recent research [47]. The GAN can be categorized into generative models or unsupervised learning such as k-means, auto-encoders that can represent the features from the unsupervised perspective. From the perspective of generative models, the algorithm should learn the distribution of the data, which can be considered a class-conditional probability from the Bayes point of view. However, for complex data, such as high-resolution images, learning the distribution of its pixels is a challenging problem. Therefore, before the invention of GAN, the previous algorithm has not achieved satisfying results for generating natural images.

The GAN consists of a generator and a discriminator. In most cases, both the generator and the discriminator are deep neural networks, such as multi-layer perceptrons, or convolutional neural networks. For the GAN, after many iterations of generative adversarial training, the generator can generate a realistic image, close to the training picture, but not the same. Therefore, the task of the generator is to learn an approximate distribution of training data. Define $G(z;\theta_g)$ as the generator that has noise z as input and θ_g as its parameters. Define $D(x;\theta_d)$ that output a binary number represents its decision. Assume the input noise has the distribution $p_z(z)$ and the generator has the distribution p_g. D(x) stands for the probability that x is from the real distribution rather than the generator's distribution p_g. Firstly, train D to maximize the ability to distinguish an input data is from distribution p_g or p_x. Then train G to minimize $log(1 - D(G(z)))$ to make D unable to distinguish the generated data is from p_g or p_x. The two steps are training iteratively for many epochs. The minimax game that G and D played can be expressed in the following mathematical formula:

$$\min_G \max_D \quad V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[logD(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[log\left(1 - D\big(G(\boldsymbol{z})\big)\right)\right] \qquad (2.32)$$

The adversarial process between G and D can be considered as a Game Theory between the counterfeit coin manufacturer and the cop. The cop (D) constantly learns the difference between

counterfeit and real money to prevent counterfeit coin makers (G), and counterfeit coin makers continue to learn the appearance of real money, making counterfeit money more realistic to deceive the cops. This situation can be modeled mathematically using minimax game in Game Theory. GAN simulates the adversarial process by using two neural networks as the counterfeit coin manufacturers and the cops. By training the two networks iteratively, the identification ability of the two networks keeps increasing.

Table 2-4. The mini batch stochastic gradient descent algorithm for training a generative adversarial nets [47].

Table 2-4 Mini batch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyper parameter. We used k=1, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample mini batch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g{}^{(z)}$
- Sample mini batch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$$

end for

- Sample mini batch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

2.6. Deep Learning Frameworks

Deep learning requires many calculations. It usually contains a neural network with many nodes, and each node has many connections that need to be constantly updated during the learning process. To develop a deep learning program requires good coordinates of the data flow and resources such as GPU and CPU. In light of this, many big companies developed robust deep learning frameworks to support and ease their research. These frameworks make it easier for data scientists and engineers to build deep learning solutions for complex problems and perform more complex tasks.

2.6.1. NVIDIA Driver and CUDA Toolkit

For this thesis, all the experiments are trained on two NVIDIA GTX1080 video cards which have the computation ability 6.1. In order to move the intensive computation from CPU to GPU and make full use of the GPU parallel computing. The NVIDIA drive 390.87 was installed together with CUDA 8.0 toolkit, and the cuDNN 7.0 was installed to further supplement CUDA toolkit. The whole project was implemented with Python3.

2.6.2. TensorFlow

TensorFlow, initially developed by researchers and engineers at the Google Brain Team, is an open source software library for numerical calculations using data flow graphs [48]. Its purpose is to accelerate deep neural networks and machine intelligence research. Since the end of 2015, TensorFlow's library has been officially open sourced on GitHub. TensorFlow is very useful for quickly performing graphics-based calculations. The flexible TensorFlow API can deploy models between multiple devices through its GPU-enabled architecture. TensorFlow currently has a large active community and rich low-level and high-level API. TensorBoard is the visualization tool for TensorFlow. TensorFlow does not only supports deep learning but also supports reinforcement learning and other algorithms. However, the drawback of TensorFlow is that it redefined all the APIs and not python friendly, which requires developers to spend more time to learn.

Since the computation is based on the static data flow graph, we first need to create a data flow graph, and then feed our data into the data flow graph. Nodes in the diagram are generally used to represent the mathematical operations, but can also represent the end of the data flow or the start of the data flow. The edge that connects two nodes represents the multidimensional data arrays, which is called tensor. During training, the tensor will continuously flow from one node in the data flow graph to another node. Once all the tensors at the input are ready, the nodes will be assigned to various computing devices to perform the operations asynchronously in parallel.

2.6.3. PyTorch

PyTorch is very popular among academic researchers and is a relatively new deep learning framework. The Facebook Artificial Intelligence Research Group developed PyTorch to address some of the problems encountered in its predecessor, Torch [49]. Due to the low popularity of the programming language Lua, Torch can never experience the rapid development of Google

TensorFlow. As a result, PyTorch uses the original Python imperative programming style that is already familiar to many researchers, developers, and data scientists. It also supports dynamic computational graphs, a feature that makes it attractive to researchers and engineers working on time series and natural language processing data. In this thesis, PyTorch (version 3.1 and 4.1) is chosen as our only deep learning framework tool.

For all the experiments, the Visdom package is used as the visualization tool. Visdom is developed by Facebook, aims at remote data visualization and scientific experiments. Visdom has different environments for different projects. For each environment, the user interface starts with a blank panel; it can be filled with charts, images, and text. The figures and charts are shown in the window of the panel. The content that appears in the window can be dragged, dropped, resized and destroyed. It is more flexible than the TensorBoard that provided by TensorFlow.

CHAPTER 3. METHODOLOGY

In this chapter, we start discussing the experiment details. First, we will discuss the dataset information that we used in this experiment. Next, we introduce the details of the implementation for the nucleus segmentation, lung segmentation, fetal brain segmentation, and the pneumonia localization. Finally, we discuss the evaluation criteria of the model.

3.1. Data Source

3.1.1. Source

In this thesis, the experiments are mainly based on three datasets, which are Data Science Bowl 2018 nucleus dataset, RSNA Pneumonia Detection Challenge dataset, and fetal brain dataset.

3.1.1.1. Nucleus Dataset

There are around 670 nuclei images and corresponding instance masks on the nucleus dataset. The dataset is acquired from the different database that is acquired under different conditions so the features of the image such as cell type, size or image modality varies. The U-Net and Mask R-CNN model are chosen to perform the semantic segmentation and instance segmentation, respectively.

3.1.1.2. Pneumonia Dataset

There are around 25000 2D single channel CT images on the pneumonia dataset that provided in DICOM format. I applied U-Net to perform a lung segmentation and feed the segmented results into the ResNet and DenseNet to perform a classification task to distinguish the normal lung, cancer lung and the pneumonia lung. Finally, a Faster R-CNN model is trained to predict the bounding box of the pneumonia area with a confidence score.

3.1.1.3. Fetal Brain Dataset

The fetus MRI dataset provided by the corresponding hospital consists of 19 subjects for the gestational age (GA) 30-33 weeks. Three scan types, coronal, transverse and sagittal, are performed on each subject that generated several volumes of MRI scans. There are 2139 images, 1900 images, and 2669 images for coronal, transverse and sagittal, respectively. All the images

are provided in DICOM format without corresponding masks. In order to perform the segmentation tasks on the fetus brain, MATLAB ground truth labeler toolbox is chosen to label all the images manually.

3.1.2. Preprocess

The data preparation for the nucleus dataset includes resizing, random crop and random flip. Since the original images are in different dimensions but the U-Net model requires the same input dimension, all the images are resized to 256x256 pixels. Because of the limited number of training images (670), the random crop and random flip data augmentation techniques are applied to the existing images to extend the number of training samples. The Caffe normalization is applied to each image before feeding into the network.

For the pneumonia dataset, the histogram equalization technique is used to enhance the contrast of the CT image for better classification. For the bounding box prediction, before the images are feed into the feature extractor, re-scale them such that their shorter side equals 600 pixels or longer side equals 1000 pixels, at least one requirement should meet, then apply the same re-scale to their corresponding bounding boxes. Then the Caffe normalization is implemented to make the neural network converge faster. In Caffe normalization, the channel of the raw image is converted from RGB to BGR, for each channel, the values are minus 122.7717, 115.9465 and 102.9801 respectively. This will make the pixel value between around -125 to 125 and having the average value 0. Finally, the images will end up having the shape (3xHxW) which stands for 3 BGR channels, height, and width. The format of the bounding box will be $(y_{min}, x_{min}, y_{max}, x_{max})$, y_{min} and x_{min} can represent the upper left corner of the bounding box, y_{max}, and x_{max} can represent the bottom right corner.

3.2. Semantic Segmentation Implementation

In this section, I will discuss the details of the configuration that has been used on the semantic segmentation tasks. Due to the difference of the nucleus dataset and the pneumonia dataset, there are minor modifications for the training configuration and the U-Net models.

3.2.1. Nucleus Segmentation

Figure 3-1 shows the U-Net model we used on the nucleus segmentation. The U-Net model is slightly modified from the original U-Net model to fit the nucleus dataset better. Since the dimension of all training images is between 150 – 350 pixels, the dimension for the input layer of the U-Net is set to 256x256 pixel. The original U-Net did not use padding so the dimensional of the feature map will slightly shrink after each convolution. In order to make it easier to concatenate features, the padding is used in our implementation. Also, the batch normalization is added in each convolution layers to accelerate the convergence of the network. The modified U-Net architecture is shown below.

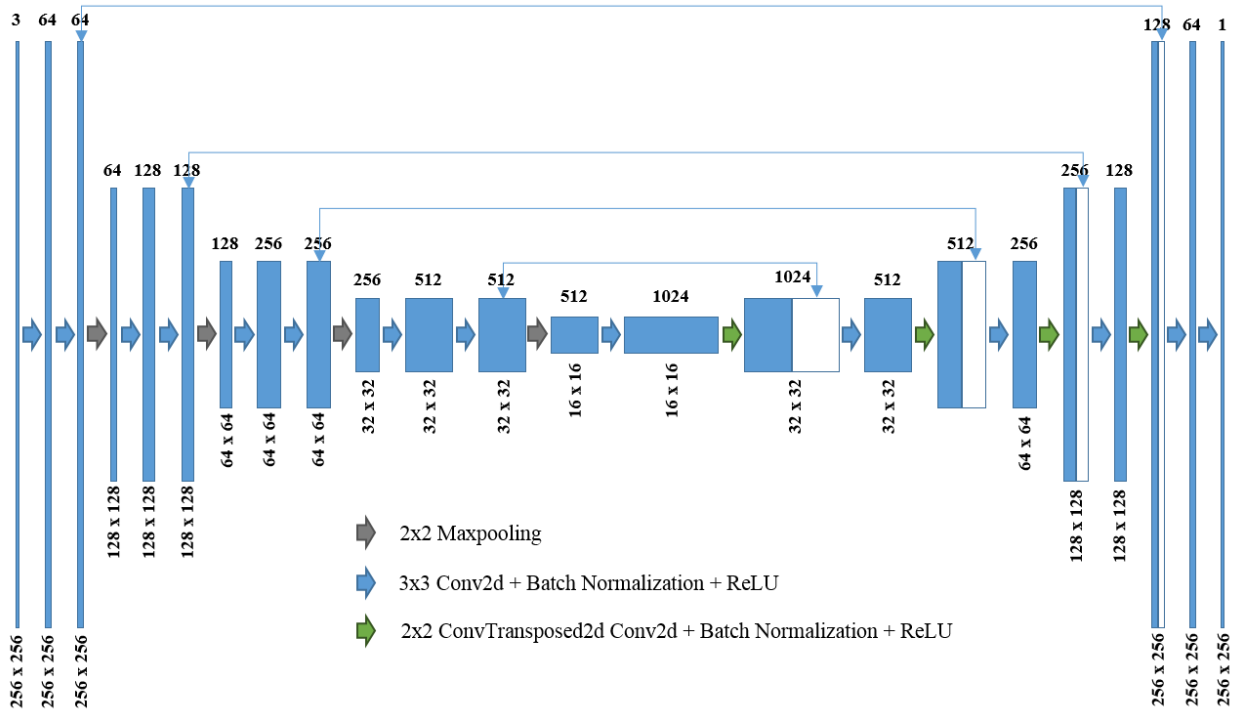

Figure 3-1. The schematic of U-Net used in nucleus segmentation lung segmentation and fetal brain segmentation.

There are many nuclei cells in each training image. The ground truth mask is separated into many images, and each image is a mask for one nucleus. Since U-Net performs the semantic segmentation, these single nucleus masks should be combined to one mask. The input image and the combined mask are resized to 256x256 pixels and feed into the U-Net. A series of feature extraction is performed and followed by feature concatenation to restore the loss of spatial information. Finally, the dimension of the feature is restored to the same as the input image, so the

binary cross-entropy (BCELoss) evaluation criteria can be applied to the input image and the output prediction. The Adam optimizer is used with learning rate = 1e-3, momentum=0.9 and weight decay=1e-4.

3.2.2. Lung Segmentation

In order to remove the unnecessary features from the CT image and only keep the lung area, a U-Net model is implemented to segment the lung out from the CT image. In order to create the dataset, the contour of 1000 CT images are manually labeled and used these images as the training set to train a U-Net. I hope this will improve the accuracy of pneumonia classification. In this experiment, the original CT images, which are 1024x1024 pixels, are resized to 512x512 pixels. The schematic of the U-Net model I used is shown below.

A batch of single-channel 512x512 images is feed into the network, followed by a series of CNN layers to extract the features. The blue arrow represents a CNN block, which is the combination of a convolution layer, batch normalization layer and ReLU layer. The kernel of the convolution layer has the size 3x3, stride 2, and zero padding. The double-arrow denotes the feature concatenation. Finally, a batch of 512x512x1 probability matrix is output to represent the segmented image. The binary cross-entropy loss is calculated between the input image and the output prediction. The Adam optimizer is used with learning rate 1e-3 and weight decay 1e-4. Since the vast amount of parameters in U-Net, the model is parallelized in two NVIDIA GTX 1080 video cards with eight images for one batch.

3.2.3. Fetus Brain Segmentation

Fetal MRI has been increasingly used in quantitative brain development studies as well as normal/abnormal development diagnosis for the gestational age (GA) 30-33 weeks [50]. For each MRI scan, the MRI scans slowly from the top slice to the bottom slice and generate a volume of images, usually, 20-60 slices depending on the scan direction. While fast imaging methods are capable of acquiring in-plane motion free 2D snapshot, unconstrained fetal motion in uterus between slice acquisitions usually leads to unregistered image volumes. Recent advances show promising volume reconstruction results from a stack of unregistered images through fetal image motion correction and super-resolution volume reconstruction. The performance of most techniques in 3D volume registration reconstruction usually depends on the accuracy of fetal brain

segmentation [51], [52]. In this study, a customized U-Net [46] and a customized FPN network [42] were implemented for automatic fetal brain segmentation. The results were evaluated by commonly used measures in medical image segmentation, and compared with FSL's popular brain extraction tool (BET). The algorithm was tested on both sagittal data (SAG), transverse data (TRA) and coronal data (COR) for 30-33 weeks gestational age.

Firstly, a modified U-Net model was implemented to fit the fetal brain segmentation task better. The architecture of the model is shown in Figure 3-1, which is the same as the model for lung segmentation. The dimension for the input layer was resized from 640×640 to 512×512 pixels for convenient feature concatenation in implementation. The convolution was set to have the kernel size of 3×3, stride 2, and zero padding. The batch normalization was adopted for faster and more stable network convergence. Manually segmented images in 52 volumes from 19 fetuses with GA between 30-33 weeks were served as the training dataset. The binary cross-entropy loss was calculated between the ground truth and the output prediction. The SGD optimizer was set to a learning rate of 0.01 with a decay rate of 50% for every ten epochs. The network converged, and the loss started to plateau after 20 epochs. The training time was approximately 5 hours on a computer with two NVIDIA GeForce GTX 1080 video cards. The reason why the Adam optimizer was replaced with SGD in this experiment is that we found that SGD converges faster than the Adam optimizer and achieves better results, the comparison between SGD and Adam training is shown in Figure 3-2.

Except for the U-Net, we also constructed a customized FPN by combining the ResNet as the backbone and the FPN as the head network. The architecture of the customized FPN is shown in Figure 3-3. The dimension of the input image is 1x512x512. For the bottom-up path, the image is feed into a ResNet and the features C2, C3, C4 and C5 are intermediate features from the ResNet. These features are passed into a 1x1 convolution layers. On the top-down path, the M5 is up-sampled twice and added with the 1x1 convolution of C4. The whole process is repeated until the M2 is generated. The features M2 – M5 are further passed into a 3x3 convolution layers to generate the pyramid features P2 – P5. P2 – P5 are up-sampled and concatenated to generate feature P, followed by a 3x3 convolution layer and a 1x1 convolution layer.

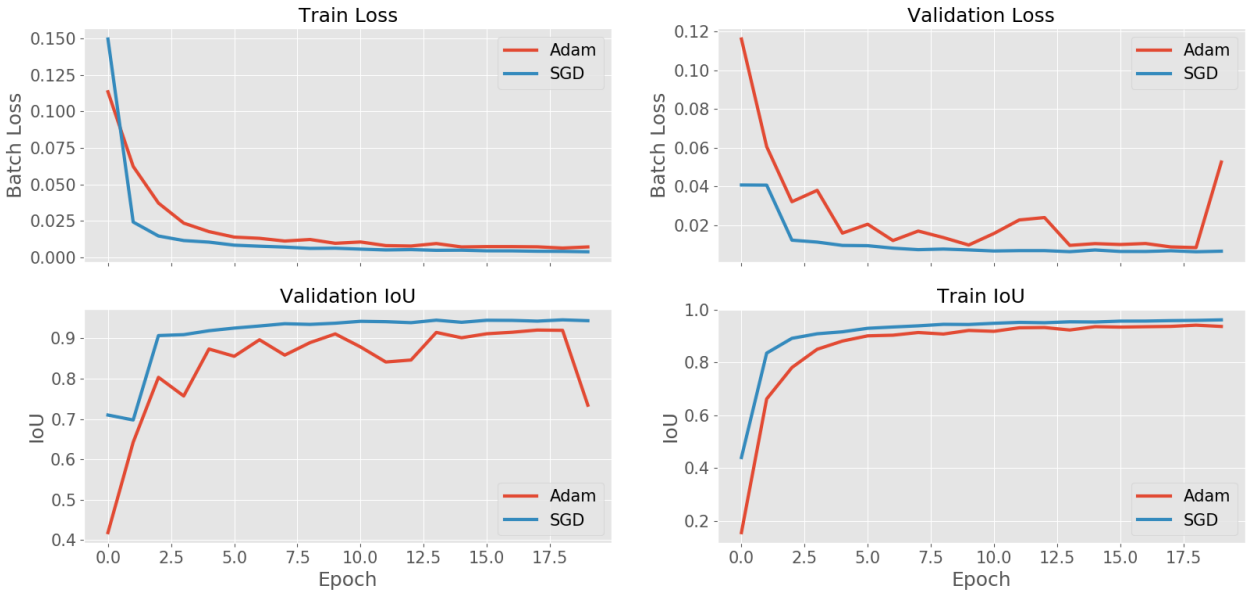

Figure 3-2. The comparison between SGD and Adam optimizer training on Fetus Brain dataset. It is shown that the U-Net converges faster and achieve better results with SGD optimizer with momentum=0.9 and learning rate=0.01.

Both the trained network and FSL's BET program were applied to the test dataset consisting of the subjects not in the training dataset for comparison. The results from FSL's BET with the fractional intensity threshold of 0.80, the best segmentation results by visual inspection with the threshold ranging from 0.5 to 0.9, were used as a reference. The performance of segmentation was evaluated by intersection over union (IOU), Dice coefficient, sensitivity and specification.

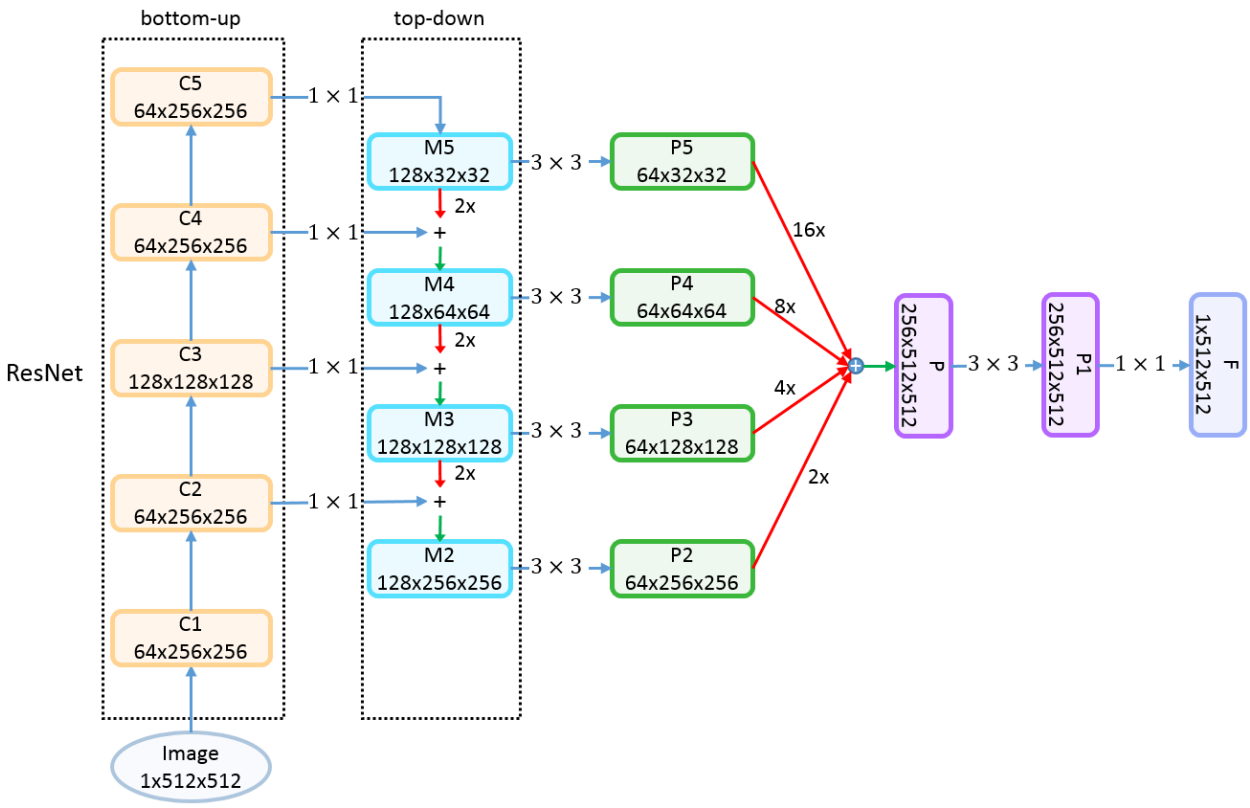

Figure 3-3. The customized feature pyramid network for semantic segmentation.

3.2.4. Fetus Brain Segmentation with GAN Loss

Fetus brain segmentation has been a challenging task in biomedical image segmentation. It requires the model output a binary mask that has the binary value for each pixel [53]. Recently, there are many FCN like models, such as U-Net or FPN, achieves very high IoU in semantic segmentation [42], [45]. However, these models all use binary cross-entropy loss function to optimize the parameters, which makes the model insensitive to the shape of the masks and predict some weird shapes such as some tiny holes in the big masks or some isolated tiny masks. These tiny holes or masks contributes little to the total loss, so even the model is trained for many epochs it still cannot fix this problem. However, for human beings, it is easy to detect the defect of the generated masks. In order to solve this problem, we combine the training of U-Net with GAN loss to boost segmentation performance [54]. The proposed schema is shown in Figure 3-4.

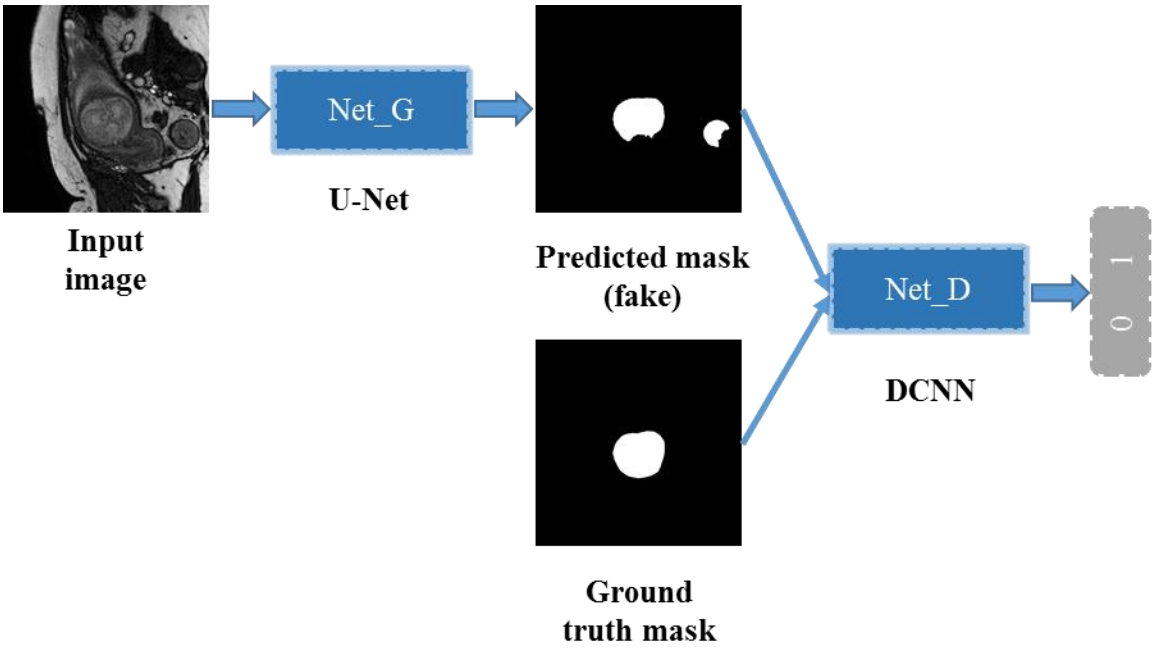

Figure 3-4. Fetus brain segmentation with GAN loss. Generator is a U-Net model that output a probability matrix that represents the segmentation mask. Discriminator is a deep convolutional neural network that classify the predicted mask and the ground truth mask.

A discriminator Net_D is adopted to classify the predicted mask and the ground truth mask. Except for the traditional binary cross-entropy between the predicted mask and the ground truth mask, the additional loss function is added to guide generator Net_G, which is a U-Net model, to predict more realistic and reasonable masks. For the discriminator $D(Y;\theta^D)$, the output probability will compare with the Y_{gt} and $Y_{pred} = G(X;\theta^G)$ which are assigned 1 and 0, respectively. The loss function for updating discriminator is:

$$\begin{aligned} l_D &= -\mathbb{E}_{y\sim p_{gt}} log\big(D(y;\theta^D)\big) - \mathbb{E}_{y'\sim p_{pred}} log\big(1 - D(y';\theta^D)\big) \\ &= -\mathbb{E}_{y\sim p_{gt}} log\big(D(y;\theta^D)\big) - \mathbb{E}_{x\sim p_{data}} log\big(1 - D(G(x;\theta^G);\theta^D)\big) \end{aligned} \tag{3.1}$$

During the training process, the generator and the discriminator are trained alternatively. Firstly, train the discriminator using the generated image and the real image. Then train the generator using the following loss function. The $l_G$consists of two type of loss, one is the traditional binary cross-entropy between the predicted mask and the real mask. The other is the GAN loss.

$$
\begin{aligned}
l_G &= \mathbb{E}_{y\sim p_{pred}, y'\sim p_{gt}}[l_{seg}(y,y')] - \lambda \mathbb{E}_{y\sim p_{pred}} log\big(1 - D(y;\theta^D)\big) \\
&= \mathbb{E}_{y\sim p_{pred}, y'\sim p_{gt}}[l_{seg}(y,y')] - \lambda \mathbb{E}_{y\sim p_{data}} log\big(1 - D(G(y;\theta^G);\theta^D)\big) \\
&= \mathbb{E}_{y\sim p_{pred}, y'\sim p_{gt}}\big[l_{seg}(y,y')\big] - \lambda \mathbb{E}_{x\sim p_{data}} logD(G(x;\theta^G);\theta^D)
\end{aligned} \tag{3.2}
$$

The updating algorithm is shown below.

Table 3-1. The Adversarial training for fetal brain segmentation task [54].

Table 3-1 Adversarial training of generator and discriminator.

Input: pre-trained generator customized U-Net with weights θ_0^G
Output: updated generator weights θ_1^G

for number of training iterations **do**
  for k_D steps do
    sample a mini-batch of training images $x\sim p_{data}$;
    generate prediction y_{pred} for x with $G(x;\theta_0^G)$;
    $\theta^D \leftarrow$ propagate back the stochastic gradient $\nabla l_D(y_{gt}, y_{pred})$;
  end for
  for k_G steps **do**
    sample a mini-batch of training images $x'\sim p_{data}$;
    generate y'_{pred} for x' with $\mathrm{G}(x';\theta_0^G)$ and compute $\mathrm{D}(G(x'))$;
    $\theta_1^G \leftarrow$ propagate back the stochastic gradient $\nabla l_G(y'_{gt}, y'_{pred})$;
  end for
  $\theta_0^G = \theta_1^G$
end for

3.3. Object Localization Implementation

From the programming perspective, the pipeline of Faster R-CNN can be divided into three parts, which is feature extraction followed by Region Proposal Network followed by object type classification and bounding box coordinates regression. The prepared dataset is feed into the feature extraction layer which is usually a VGG network or ResNet network, then the feature map of the input image is generated. The region proposal network is responsible for finding the region of interests (ROI), which indicates the area that may contain an object, from the feature map. After the ROI is located, feed both feature map and ROI into Faster R-CNN core network to perform an

object classification and bounding box regression. The pipeline of Faster R-CNN is shown in Figure 3-5.

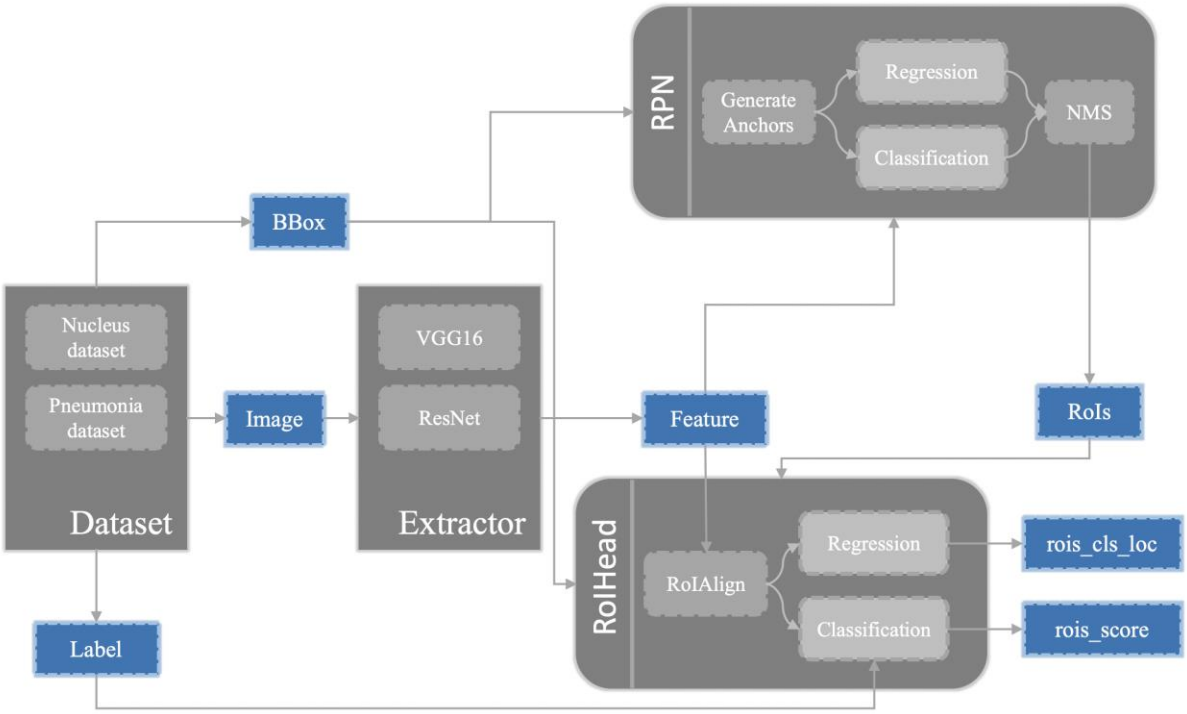

Figure 3-5. Pipeline of the Faster R-CNN architecture

3.3.1. Extractor

For the experiment in this project, the Caffe pre-trained very deep VGG-16 model is used to perform a feature extraction task. All layers are initialized by Caffe pre-trained model for ImageNet classification, which is a common practice for transfer learning. Then the learning rate of the first four convolution layers is set to zero.

The VGG consists of two parts, one is 13 convolution layers for feature extraction, and another one is three fully connected layers for classification. The 13 convolution layers are used in Faster R-CNN extractor, and the first two of the three fully connected layers are used in Faster R-CNN cores.

Because of the four pools between the input image and the output features, the spatial dimension is downsampled 16 times, which means if the input image is in size (3, H, W), then the output

feature will be in size (C, H/16, W/16), where the C stands for the output channels from the last convolution layer.

3.3.2. Selective Search

The selective search employees the bottom-up hierarchical grouping algorithm that can continuously generate different scales of locations and group the adjacent areas until the different small regions become a whole image. The method of Felzenszwalb and Huttenlocher [55] are used as the start point to generate small regions. Then the greedy algorithm is used iteratively to group the adjacent regions. The first step is to calculate the similarity between the adjacent areas. If the similarity of the two regions is greater than a threshold, then group them to one region. The following pseudo code illustrates the details of the algorithm.

3.3.3. Region Proposal Network

The Region Proposal Network, an innovative idea in Faster R-CNN, is responsible for proposing candidate object bounding box, which employs the "attention" mechanisms to tell the network where to look [4]. Compare with traditional low-level feature merge based selective search, the Region Proposal Network reduces the time for candidate object generation to almost real time (from 2s to 0.01s) without affecting the final accuracy.

The anchor, the fixed size bounding boxes, is a very a fundamental concept in Faster R-CNN. When the feature map come from the extractor and feed into Region Proposal Network, a sliding window moves on the feature map and generates different anchors in different scales and aspect ratios in the corresponding area of the input image. We follow the default setting in the Faster R-CNN paper that use three scales ($128^2, 256^2, 512^2$) and three aspect ratios (1:1, 1:2, 2:1) that generating nine anchors at each sliding position in the feature map. We end up having H*W*9 anchors for each training image, the H and W stands for the height and width of the feature map. For example, if the size of the feature map is 512x50x60, it will generate 50x60x9=18000 anchors. For each sliding window position, an intermediate fully convolutional layer is added followed by two parallel fully convolutional layers that produce 18 scores and 36 coordinates which stands for the possibility that there is an object and the bounding box of this object. Please note that the intermediate layer, classification layer, and regression layer are the fully convolutional network,

which means there is no requirement for the dimension of the feature because the kernel size is 1x1 that does not change the shape of the feature map.

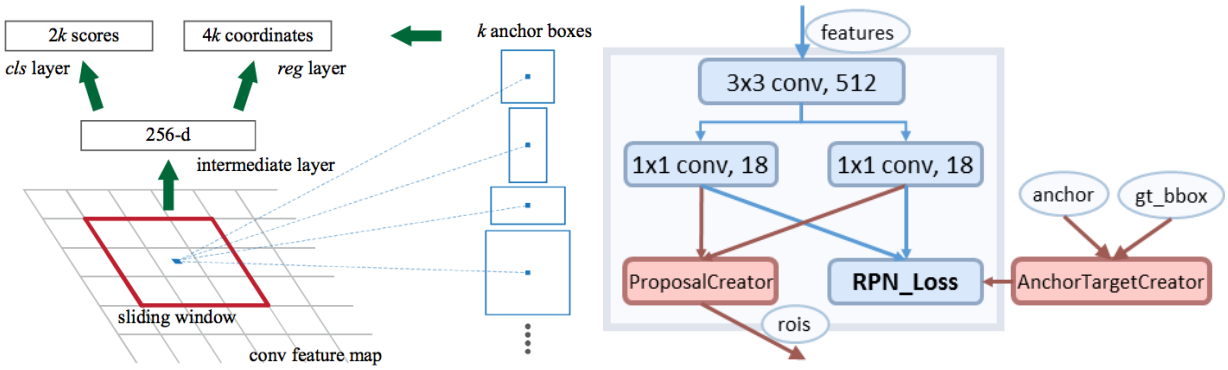

Figure 3-6. The RPN architecture.

However, the number of anchors, which is around 20000, is too large. If passing all the anchors into the classification and regression layer, it will significantly slow down the training process. Here, what we usually do is filter 256 anchors from these 20000 candidate anchors by a set of rules and only use these 256 anchors as the training set to train the intermediate layer, classification and regression layer. The filter process is the following:

Choose the anchors that have the highest Intersection-over-Union (IoU) with each ground truth bounding box as positive samples.

Randomly choose the anchors that have the IoU larger than 0.7 with any ground truth bounding box from the rest of the anchors as positive samples. The total number of positive samples should no more than 128.

Randomly choose some anchors that have IoU less than 0.3 with any ground truth bounding box as negative samples. The total number of positive samples and negative samples should be 256. The IoU is the parameter to measure the ratio of intersection between the predicted box and ground truth box. The formula for computing IoU is:

$$IoU = \frac{Predicted \cap GroundTruth}{Predicted \cup GroundTruth} \tag{3.3}$$

Where $Predicted \cap GroundTruth$ represent the overlap area between predicted bounding box and ground truth bounding box, and $Predicted \cup GroundTruth$ stands for the union area of predicted bounding box and the ground truth bounding box.

For the 256 training anchors, label them either 1 or 0, which stands for there is an object (foreground) and there is no object (background). Instead of directly using (y_1, x_1, y_2, x_2) coordinates as the label of the bounding box, we follow the Faster R-CNN paper that use the following (t_x, t_w, t_x^*, t_w^*) as the label of bounding box:

$$
\begin{aligned}
t_x = \frac{x - x_a}{w_a}, t_y = \frac{y - y_a}{h_a} \\
t_w = log\frac{w}{w_a}, t_h = log\frac{h}{h_a} \\
t_x^* = \frac{x^* - x_a}{w_a}, t_y^* = \frac{y^* - y_a}{h_a} \\
t_w^* = log\frac{x^*}{w_a}, t_h^* = \frac{h^*}{h_a}
\end{aligned}
\tag{3.4}
$$

The x, y, w and h stand for center coordinates, width and height of the bounding box. The variable x, x_a and x^*represent the predicted bounding box, anchor box and ground truth bounding box respectively.

The function of the RPN network is to tell the Faster R-CNN core network where there might be an object in the image, then the Faster R-CNN core network crops the corresponding feature map to perform the classification and bounding box regression task. Thus, during the RPN training process, it also generates the RoI for training the Faster R-CNN core network. The RoI generating process is in the following.

- For each image (C, H, W), perform a forward pass on extractor and RPN network to calculate the probability that the H/16 * W/16 * 9 anchors boxes having an object and the location of the regressed bounding box.
- Choose 6000 anchors that having higher object probabilities from the H/16 * W/16 * 9 anchors boxes.

- Use the result of bounding box regression, adjust the coordinates of the 6000 anchors to get the 6000 RoIs.
- Use Non-maximum suppression (NMS) to choose 300 RoIs that have higher probability from the 6000 RoIs.

Please note that the progress of generating the RoIs only perform a forward pass, only training the RPN requires a backward pass.

3.3.4. Non-Maximum Suppression

One of the problems for Faster R-CNN is that it may detect one object for more than once [4], [56]. Non-Maximum Suppression algorithm can suppress the detection times on the same object to make Faster R-CNN only detect an object once. Considering there are so many anchors, an object may be included in several adjacent anchors, so all these anchors will predict that there is an object. Concretely, what the NMS does is that it will look the predicted probability from these anchors and pick the anchors that have the highest predicted probability for each category. For example, if the detection problem is only detecting cars. After running the detection algorithm, it may predict the several bounding boxes and its corresponding probability on the same car. In the NMS algorithm, it will find the largest probability which is 0.9 in the following image and suppress the rest of the bounding boxes that have IoU higher than a threshold (usually 0.3-0.5). So, the two bounding boxes with probability 0.6 and 0.7 are suppressed. Then repeat the same process on other bounding boxes. In this case, there is only one category which is the car in the scenario. If there are more than one categories in the scenario, for example, cars, traffic lights, pedestrians, then the NMS should be applied to the three categories independently.

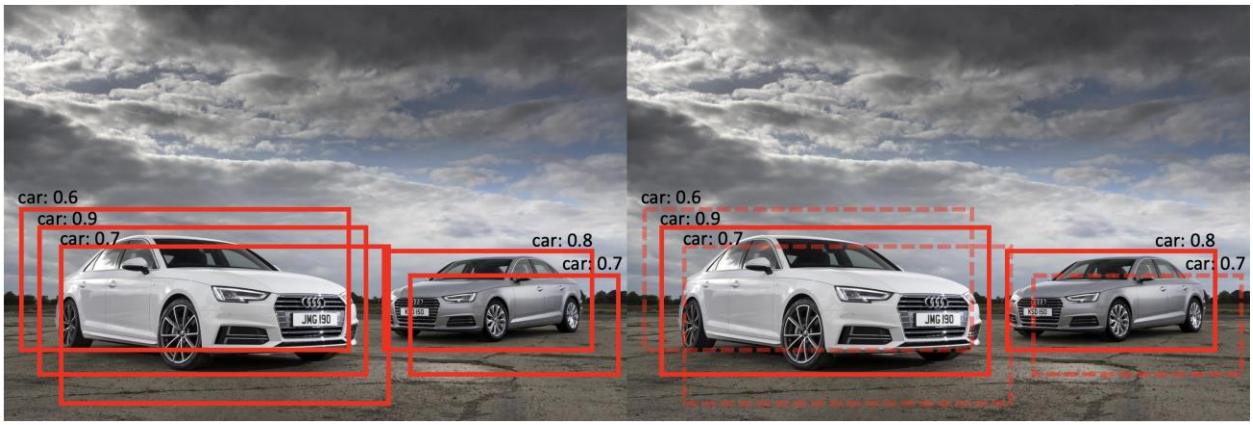

Figure 3-7. Example of how NMS works

The algorithm of NMS is in the following steps:

1. For each category, do the following steps.
2. Discard all bounding boxes with predicted probability less or equal to a threshold (0.6).
3. For the remaining bounding boxes, pick the box with the largest predicted probability and output that as a prediction.
4. Discard any remaining box with IoU greater or equal to 0.5 with the box output in step 3, and repeat step 1 until there is no bounding boxes need to be discarded.

3.3.5. Receptive Field

When the kernel moves on the feature map, for each position on the feature map, nine anchors will be generated on the corresponding original image. However, how do we determine the coordinates of the anchors on the original image? In the Convolutional Neural Network, the receptive field determines the input area size on the original image that the pixels on the feature map output by each layer of the convolutional neural network are mapped [45].

The receptive field can be calculated from the stride, kernel size and padding parameters. The receptive field of the first convolutional layer feature map equals the filter size. The receptive field of any intermediate convolutional layer feature map determined by all filter size and stride of previous convolution layers. When calculating the receptive field, no need to consider the padding size.

The simplest way to calculate the receptive field is:

- For each layer, add floor (F/2) pixels padding, the F is the kernel size. For example, if the filter size is 5, then add 2 pixel padding.
- For any feature map, the center pixel has a receptive field that centered at (0, 0) on the corresponding image.
- For any feature map, the pixel (x, y) on the feature map has a receptive field at (S_x, S_y) on the corresponding image (S is the stride of the kernel).

Once the receptive field can be calculated, when coordinates of the anchors on the original image can be determined.

3.3.6. Faster R-CNN Core Network

As discussed above, the function of RPN is pretty much the same as the selective search that tells the Faster R-CNN core network where to look at. The only difference is that the RPN achieves almost real-time RoIs generation and improves the overall detection accuracy (from mAP 39.3 to 42.1 on COCO test-dev dataset) for Faster R-CNN core network. So, the RPN only give 300 RoIs, and the Faster R-CNN Core Network takes these RoIs to perform an object classification and bounding box regression. The architecture of the Faster R-CNN core network is shown in Figure 3-8.

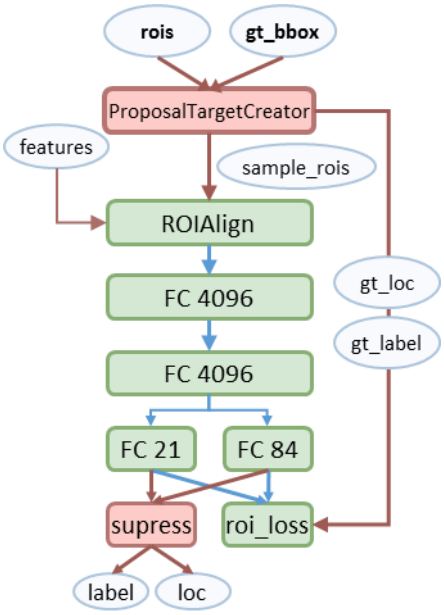

Figure 3-8. The Faster R-CNN core network architecture

Since the RoIs from RPN are in different dimensions, RoI pooling plays a critical role in reshaping the RoIs into the same dimension and feed them into a fully connected network [3], [4]. The RoI pooling technique is firstly proposed in paper Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition that makes fixed input image size not a requirement for image classification and object detection problem[57]. It is originally called spatial pyramid pooling layer. One problem for RoIPool is that it quantizes the float number of dimensions to a discrete integer which means some spatial information of the feature map is lost. For example, an 800x800 pixel image will be converted to a 25x25 (800/32=25) pixel feature map after feeding into the VGG16. The bounding box of the dog, which has the 665x665 pixel dimension, should have the dimension 20.78x20.78 (665/32=20.78). However, because of the quantization, this float number is rounded

to 20x20, so 0.75-pixel information on the feature map is lost, which means 0.75x32=24 pixels information on the original image is lost. Then, when feeding the 20x20 feature map into the RoI Pooling layer, another quantization is performed. Since the RoI Pooling layer will convert any dimension feature map to a 7x7 pixel feature map, only 14x14 pixels of the 20x20 pixels will be taken account to perform the max pooling because $20/7 = 2.86 \approx 2$. So the RoIPool has two quantization operation, the first one is from the coordinate on the image to the coordinate on the feature map, the second one is from the coordinate on the feature map to the coordinate on the RoI feature.

To avoid such information loss caused by quantization, the RoIAlign that proposed in Mask R-CNN paper uses bilinear interpolation to calculate the pixel value to avoid the quantization problem [5]. RoIAlign is proved to improve the mask accuracy by 10% to 50%. In RoIAlign, if the bounding box coordinates of the feature map are float, the exact pixel value of four corners of the bounding box is calculated by bi-linear interpolation. In the experiments we did, we use RoIAlign to achieve higher accuracy.

As we discussed before, the VGG consists of two parts; one is called extractor used for feature extraction; another is called classifier for classifying features into different categories. We load the pre-trained weights from Image-Net and decompose the VGG into the extractor and classifier. The extractor is used in the first stage of Faster R-CNN to extract feature map for RPN to use. The classifier is used in the first two layers of Faster R-CNN core network, followed by two parallel fully connected layers, one has 21 output dimension that is used to classify the category of the objects, another has 84 output dimension that is used to regress the bounding box for each category. Please notice that the RPN also perform the classification and regression, but in RPN this process is made by the fully convolutional network. However, in Faster R-CNN core, the classification and regression are achieved by two fully connected networks. The architecture is shown above in Figure 3-8. Since the Faster R-CNN core is also a network, training process is necessary. As we discussed in the RPN chapter, the RPN will generate around 300 RoIs. Instead of training the Faster R-CNN core on these 300 RoIs, we choose 128 RoIs for training by a set of rules:

- Choose the RoIs that have more than 0.5 IoU with ground truth bounding box as positive samples.

- Choose the RoIs that have less than 0.1 IoU with ground truth bounding box as negative samples.
- The total number of positive sample and the negative sample is 128.

In the prediction process, all the RoIs is feed into the Faster R-CNN core and perform a forward inference to calculate the classification results and bounding box regression results. Adjust the coordinates of the RoIs according to the bounding box regression results. Then apply NMS on the RoIs to remove the duplicate RoIs that have lower probability but high IoU with other RoIs.

3.3.7. RPN Loss

The RPN loss consists of two parts, one is object classification loss, and the other is bounding box coordinates regression loss [4]. The RPN loss is the sum of the two losses.

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{3.5}$$

For the object classification loss:

$$L(p_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) \tag{3.6}$$

Where i is the index of the anchors in a mini-batch which is one in our case, p_i is the predicted probability and p_i^* is the label for anchor, 1 stand for positive anchor and 0 represent negative anchor. The N_{cls} is the numbers of anchors (512) for training.

L_{cls} is the log softmax function that can compress any dimension vector **z** to another dimension **k** and make every output tensor lie in the range of (0, 1) and the sum of all elements are 1. Log softmax function is widely used in the classification problem.

$$L_{cls}(z)_j = -log\left(\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}\right) for\ j = 1, \dots, K. \tag{3.7}$$

For the bounding box coordinates regression loss:

$$L(t_i) = \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{3.8}$$

In which the λ is a constant value, N_{reg} is the number of total anchors (around 2000) and p_i^* is the label of the anchors, 1 stand for positive anchor and 0 represent negative anchor. Please notice that the negative anchors are ignored by multiplying p_i^* with $L_{reg}(t_i, t_i^*)$ because if p_i^* is 0 then the loss will be 0.

Here, instead of using L1 loss directly, the Faster R-CNN use smooth L1 loss function which defined by the following equation.

$$\begin{aligned} L_{reg} &= smooth_{L1}(t_i - t_i^*) \\ smooth_{L1}(x) &= \begin{cases} 0.5x^2 & if\ |x| < 1 \\ |x| - 0.5 & otherwise \end{cases} \end{aligned} \tag{3.9}$$

Where t_i is the predicted bounding box, and t_i^* is the ground truth bounding box. Instead of using the coordinates of the bounding box directly. The Faster R-CNN performs the regression base on the difference between t_i and t_i^*; the parameter t is defined by the following equation:

$$\begin{aligned} t_x &= \frac{x - x_a}{w_a}, t_y = \frac{y - y_a}{h_a} \\ t_w &= log\frac{w}{w_a}, t_h = log\frac{h}{h_a} \\ t_x^* &= \frac{x^* - x_a}{w_a}, t_y^* = \frac{y^* - y_a}{h_a} \\ t_w^* &= log\frac{x^*}{w_a}, t_h^* = \frac{h^*}{h_a} \end{aligned} \tag{3.10}$$

Where the x, y, w and h stand for center coordinates, width and height of the bounding box. The variable x, x_a and x^*represent the predicted bounding box, anchor box and ground truth bounding box respectively.

3.3.8. RoI Loss

During the training process of RPN, it also provides the RoIs for Faster R-CNN core to perform further classification and regression. In RPN, it only classifies if there is an object or not, but in the Faster R-CNN core, it classifies which category is the object belongs to by predicting a probability for each category. The Faster R-CNN core also regresses the bounding box position for each category. The loss function used here is the same as the one in RPN [4].

So there are totally 4 different losses in Faster R-CNN training process.

- RPN classification loss: if there is an object in the given anchor, this is a binary classification
- RPN bounding box regression loss: slightly adjust the position of the bounding box
- RoI classification loss: predict which categories the object belongs to by calculating the probability that the object belongs to each category
- RoI bounding box regression loss: further adjust the position of the bounding box base on the results from RPN

3.3.9. Focal Loss

We know that object detection algorithms can be divided into two main categories: two-stage detector and one-stage detector. The former refers to a detection algorithm that requires a region proposal like Faster RCNN and RFCN. Such algorithms can achieve high accuracy but at a slower speed. Although speed can be achieved by reducing the number of proposals or reducing the resolution of the input image, there is no qualitative improvement in speed. The latter refers to a detection algorithm similar to YOLO, SSD that does not require region proposal, direct regression, such algorithms are fast, but the accuracy is not as good as the former [41], [43]. The authors propose that the starting point of focal loss is also that one-stage detector can achieve the accuracy of the two-stage detector without affecting the original speed. The author claimed that the reason why the accuracy of the one-stage detector is lower than the two-stage detector is that of the imbalanced classes. In the object detection field, an image may generate thousands of candidate locations, but only a few of them contain objects, makes the category imbalance, and which means the number of negative samples is too large, accounting for the majority of the total loss, and it is easy to classify. So the optimization direction of the model is not what we hope. In light of the training class imbalance, a new loss function called focal loss was proposed by the FAIR team to

suppress the convergence speed of the negative samples while speeding up the convergence of the positive samples [58].

As discussed above, the cross-entropy loss can be expressed as

$$CE(p,y) = \begin{cases} -log(p), & if\ y = 1 \\ -log(1-p), & otherwise \end{cases} \tag{3.11}$$

Where y can either be 1 or 0 indicates the positive samples and negative samples. For notational convenience, introduce p_t:

$$p_t = \begin{cases} p, & if\ y = 1 \\ 1-p, & otherwise \end{cases} \tag{3.12}$$

So the cross-entropy loss can be written as

$$CE(p,y) = CE(p_t) = -log(p_t) \tag{3.13}$$

For focal loss function, a modulating term $(1-p_t)^\gamma$ is added to multiply with the cross-entropy loss, the focal loss is defined as:

$$FL(p_t) = -(1-p_t)^\gamma\, log(p_t) = -y(1-p)^\gamma\, log(p) - (1-y)p^\gamma log(1-p) \tag{3.14}$$

Where γ is the focusing parameter and should be equal or greater than 0.

3.3.10. Training Configuration

In the Faster R-CNN paper, the 4-step alternating training is employed which means train the RPN and Faster R-CNN core separately. However, the approximate joint training is implemented in our experiments, which is end-to-end and achieves faster speed. In our implementation, we perform a forward inference among the whole network and combine all the losses and perform a backward forwarding, which can update all parameters in the RPN and the Faster R-CNN core at the same time.

For each training and testing image, before feeding into the VGG extractor, it is rescaled such that their shorter side is no larger than 600 pixels or longer side is no larger than 1000 pixels. Because

of such implementation, the sizes of input images can vary. So our implementation only supports a single image for one batch size.

3.3.11. Confusion Matrix

In machine learning and statistical area, the predicted error of the classifier is usually evaluated by the error matrix, also called Confusion Matrix. Just as the name suggests, the Error Matrix is a table or a matrix that records and visualize the performance of the classifier [59]. For a binary classification problem, there are two rows and two columns in the confusion matrix, each row stands for the predicted condition, and each column represents the true condition. The true condition is the ground truth label that including the positive condition and negative condition, and the predicted condition is the prediction from the classifier, which also including positive predicted condition and negative predicted condition [60]. The prediction error is recorded by four parameters:

- True positive (TP): the number of samples that the classifier correctly predicts them as positive class.
- False positive (FP): the number of samples that the classifier incorrectly predicts them as positive class.
- False negative (FN): the number of samples that the classifier incorrectly predicts them as negative class.
- True negative (TN): the number of samples that the classifier correctly predicts them as negative class.

Table 3-2. Confusion Matrix used to evaluate the performance of classification problems.

Confusion Matrix		
	Real Positive	**Real Negative**
Predicted Positive	TP-True Positive	FP-False Positive (type I error)
Predicted Negative	FP-False Negative (type II error)	TN-True Negative

After these statistical records are generated by running the classifier on the validation set, the model can be evaluated by using the following terms:

Recall, sensitivity or true positive rate (TPR):

$$TPR = \frac{TP}{TP + FN} \tag{3.15}$$

Specificity, or true negative rate (TNR):

$$TNR = \frac{TN}{TN + FP} \tag{3.16}$$

Precision or positive predictive value (PPV):

$$PPV = \frac{TP}{TP + FP} \tag{3.17}$$

Accuracy (ACC):

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.18}$$

F1 score or Sørensen–Dice coefficient:

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{3.19}$$

A model only has a high Recall or precision does not necessary means high performance. For example, a classifier correctly predicts 99% diagnoses on the validation set for a cancer classification problem. If there are only 0.5% of patients have cancer in the training set then the 99% accuracy is not impressive at all, because if the classifier predicts nobody has cancer, then the classifier can achieve 0.5% error. In this skewed classes case, the negative samples (people do not have cancers) are far more than the positive samples (people have cancers). Hence, the F1 Score is usually used to evaluate the performance of the model in practice.

3.3.12. Mean Average Precision

Mean Average Precision (mAP) is widely used in information retrieval, image classification and object detection [61]. It is critical to figure out how it works. As we discussed above, both recall

and precision are not good enough to measure the performance of the model. Then why do not combine them? The precision-recall curve is the curve that uses recall as x-axis and precision as the y-axis. The different threshold can be set to calculate different pairs of precision and recall. The idea of Average Precision (AP) is taking the average value of the precision on all recall values, which can also be conceptually viewed as calculating the area under the curve. The mAP is just the average AP over all classes.

In our experiments, we evaluate the model by calculating the mean average precision over different IoU thresholds. For each threshold, the mAP can be calculated by the confusion matrix, which is further calculated by determining if the predicted bounding box matches the ground truth bounding box through IoU. The threshold values we choose to use ranges from 0.5 to 0.75 with a step size of 0.05, which is (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95). For example, when calculating the mAP under threshold 0.7, only if the predicted bounding box's intersection over union with a ground truth bounding box is larger than 0.7, then the predicted bounding box can be considered 'hit'. For each threshold value t, a precision value is calculated by using the following formula:

$$\frac{TP(t)}{TP(t) + FP(t) + FN(t)} \tag{3.20}$$

If the predicted bounding box matches the ground truth bounding box with higher IoU than the threshold, then a true positive is counted. If the predicted bounding box has the lower IoU than the threshold with any ground truth bounding box, then a false positive is counted. If a ground truth bounding box has lower IoU than the threshold with any predicted bounding box, then a false negative is counted. Finally, an average precision for a single image is calculated by summing up all precision values over all thresholds and divided by the length of thresholds.

$$\frac{1}{|thresholds|}\sum_{t}\frac{TP(t)}{TP(t) + FP(t) + FN(t)} \tag{3.21}$$

Finally, the mAP is calculated by summing up the precision for each image and divided by the number of images.

CHAPTER 4. EVALUATION

4.1. Nucleus Segmentation

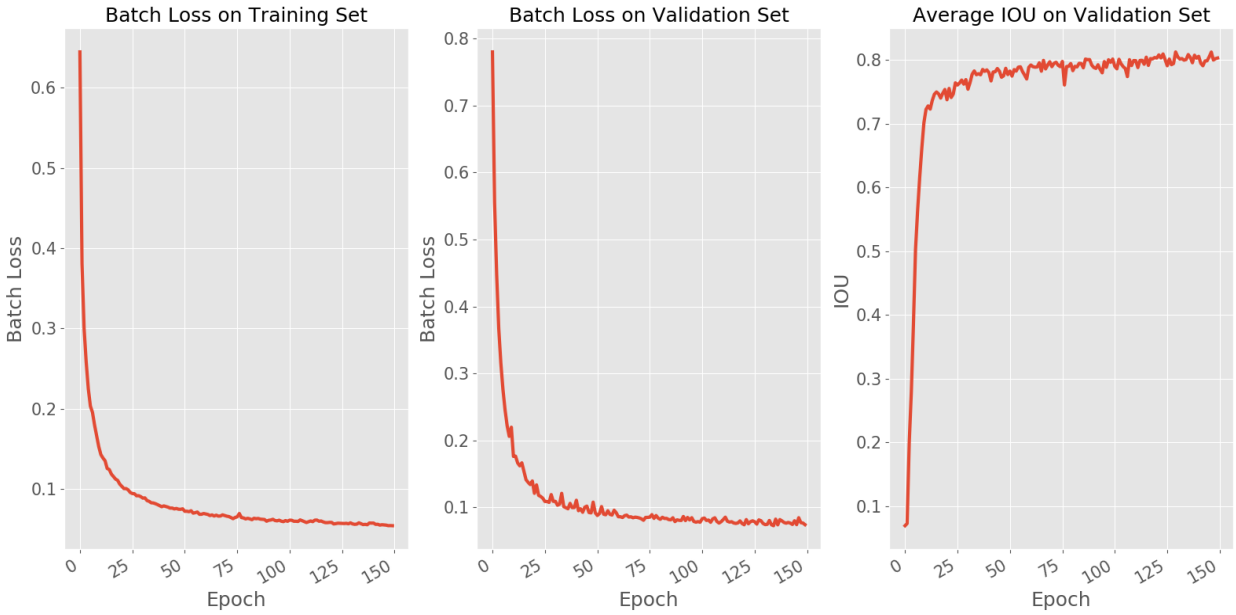

Figure 4-1. The loss on the training set (left), and validation set (middle). The average Intersect-Over-Union on validation set.

The performance of the U-Net model is evaluated on the validation set. The network converges very fast at the first 25 epochs and slows down until 75 epochs, finally plateau after 75 epochs. This might due to the simplicity of the segmentation task and the limitation of the training images. The following demos are the prediction on the validation set.

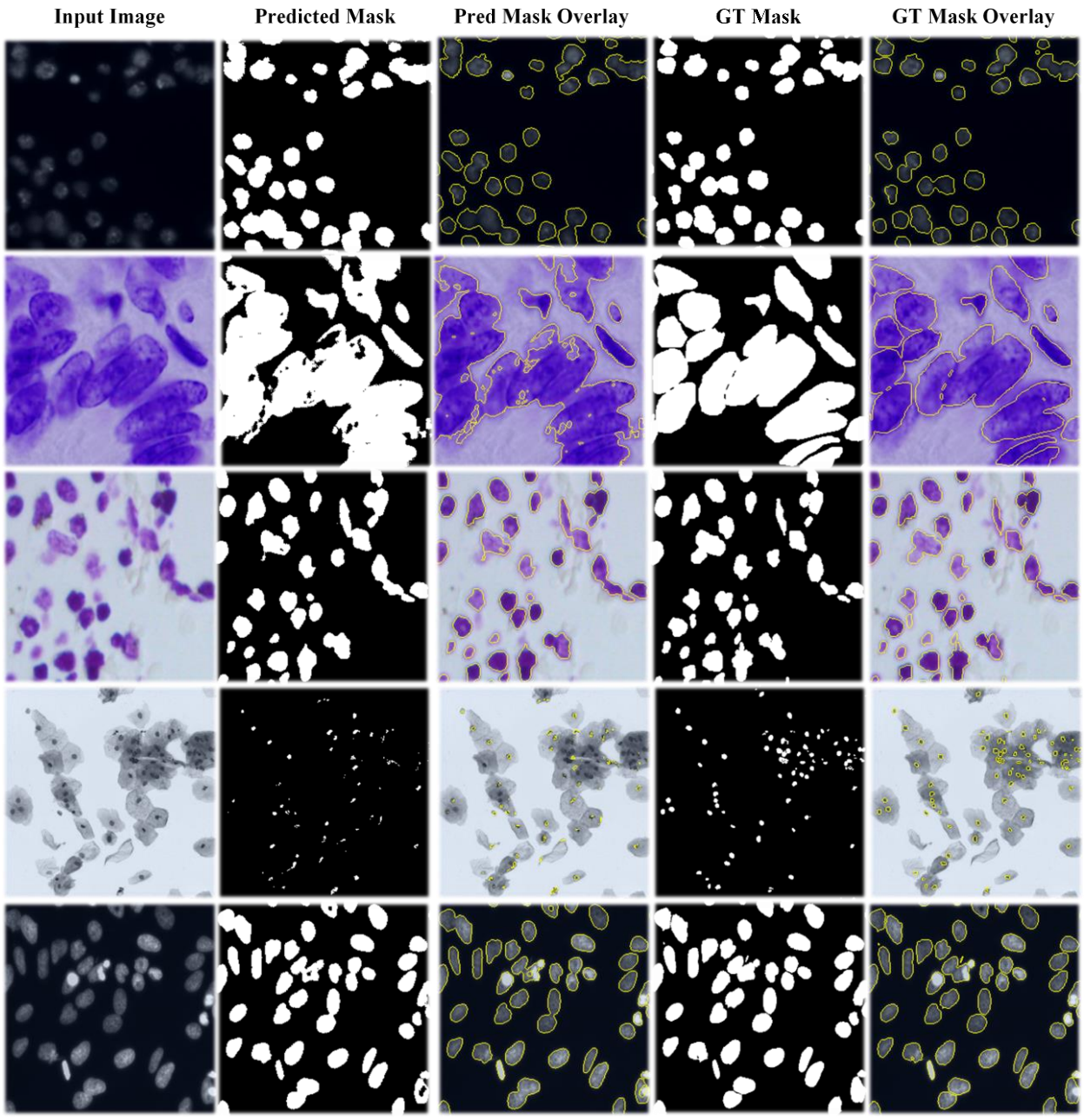

Figure 4-2. U-Net results on the validation set of the data science bowl 2018 nucleus dataset. The images are in dimension 256x256 pixels.

4.2. Pneumonia Localization

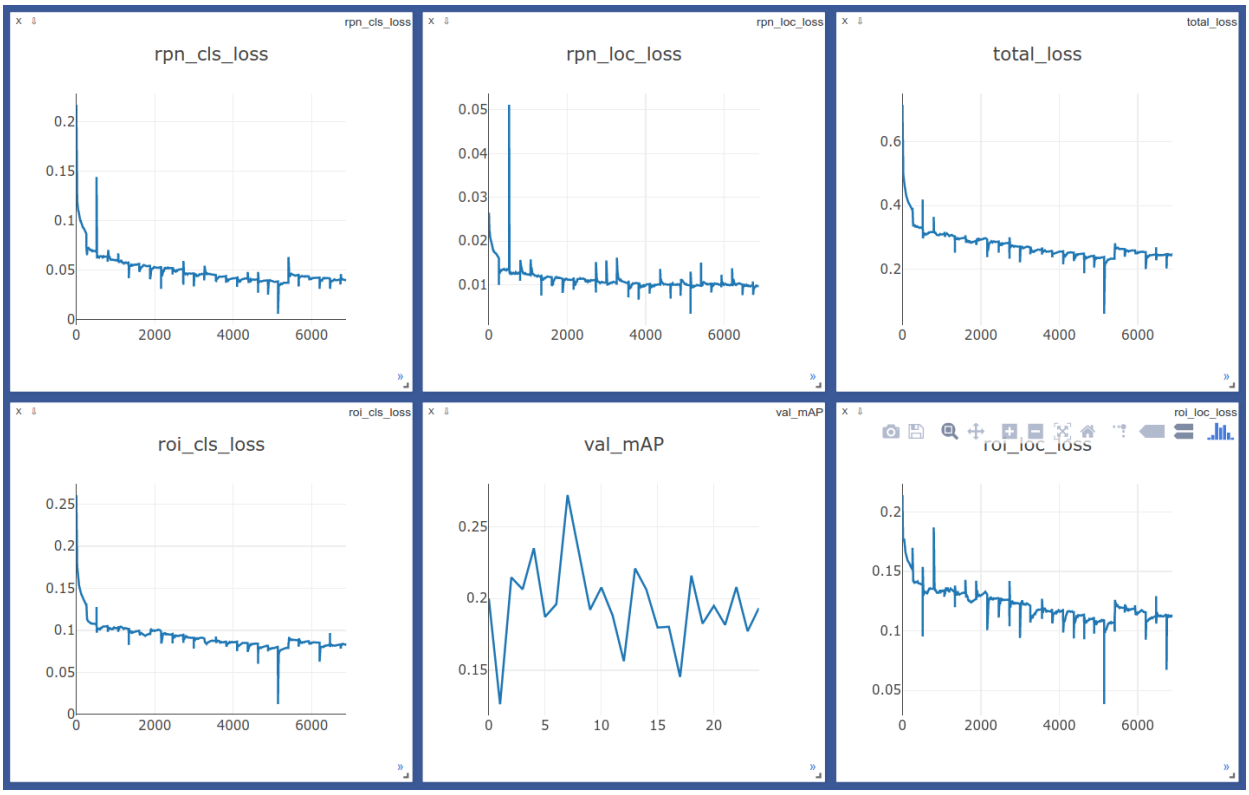

Figure 4-3. Training loss of classification on the region proposal network (top left), training loss of bounding box regression on RPN (top middle), training loss of classification on the Faster R-CNN core network (bottom left), and training loss of bounding box regression on the Faster R-CNN core network (bottom right). Total training loss of the network (top right). The mean average precision on the validation set for each epoch (bottom middle).

The 25000 CT images are split for training and validation by the ratio 9:1. All the images are read from DICOM file and resized from 1024x1024 to 600x600 and normalized by using Caffe normalization and feed into the customized Faster R-CNN that discussed in chapter 3.4. Since many subjects are healthy, there is no corresponding ground truth bounding box for them, which make the Faster R-CNN fail to utilize them. There are around 5600 samples have ground truth bounding box and 20197 samples do not have ground truth bounding box. Thus, during the training process, I discard these training samples. The pre-trained VGG16 weights on ImageNet is loaded for transfer learning. Because of the implementation method, the model only supports one training sample for each batch. The model is trained for 100 epochs and evaluated on the validation set.

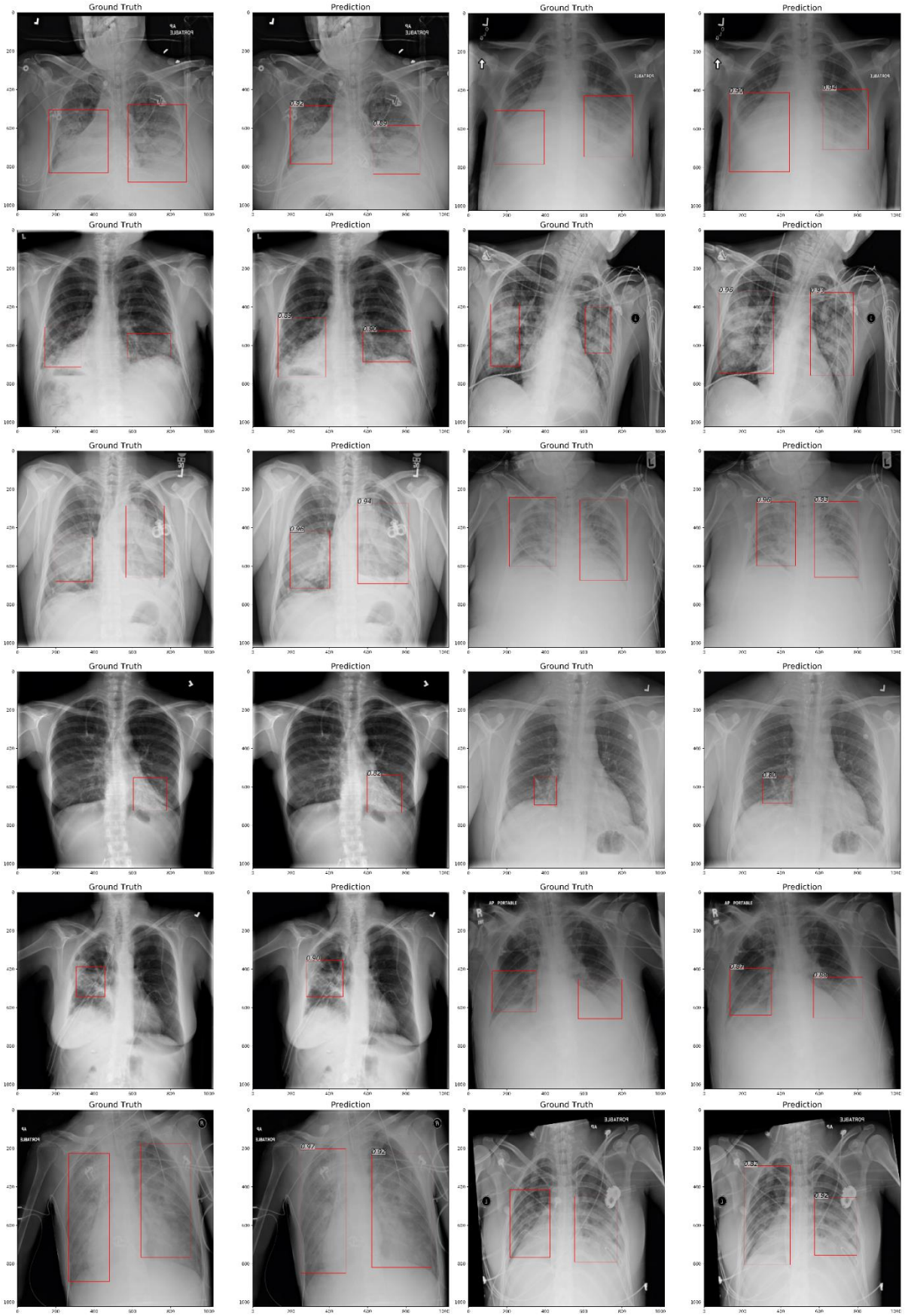

Figure 4-4. Faster R-CNN predicted bounding box along with a confidence score on the pneumonia subjects.

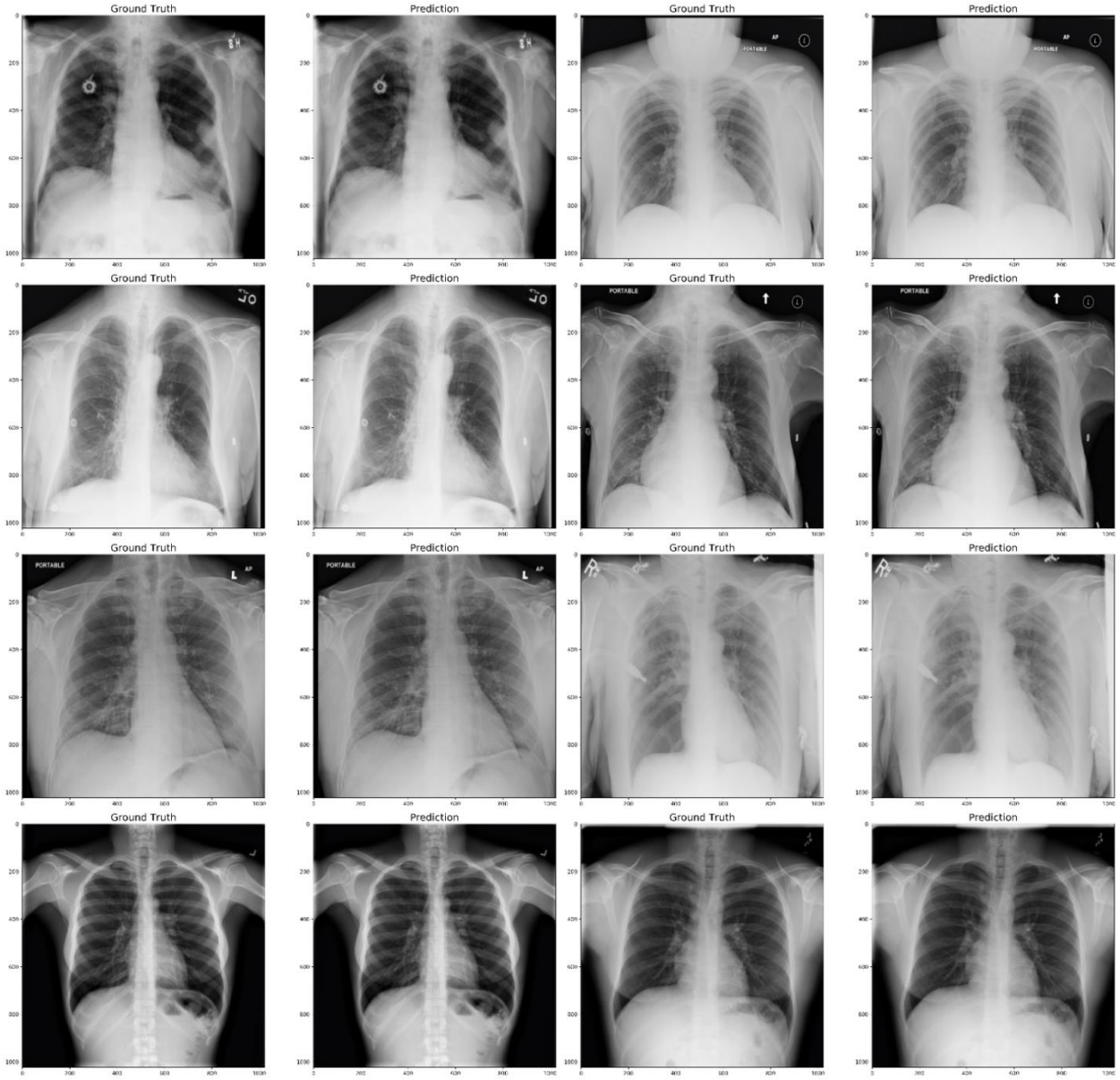

Figure 4-5. Faster R-CNN prediction on non-pneumonia subjects. Faster R-CNN did not predict anything on these subjects.

The SGD optimizer is implemented with momentum 0.9 and learning rate decay. The learning rate is set to 0.001 initially and decayed to one-tenth after every 20 epochs. The smooth L1 loss function is implemented, which is described in chapter 3.4.7 and 3.4.8. The RPN sigma and RoI sigma is set to 3 and 1 for RPN and Faster R-CNN head. Figure 4-3 shows the training loss and the performance of the trained model. The true positive predicted results are shown in Figure 4-4. The Faster R-CNN successfully predicted a bounding box along with a confidence score for the

pneumonia subjects. Figure 4-5 shows the true negative. The faster R-CNN did not predict any bounding box on the non-pneumonia subjects.

4.3. Lung Segmentation

The statistical evaluation results are shown in Figure 4-6.

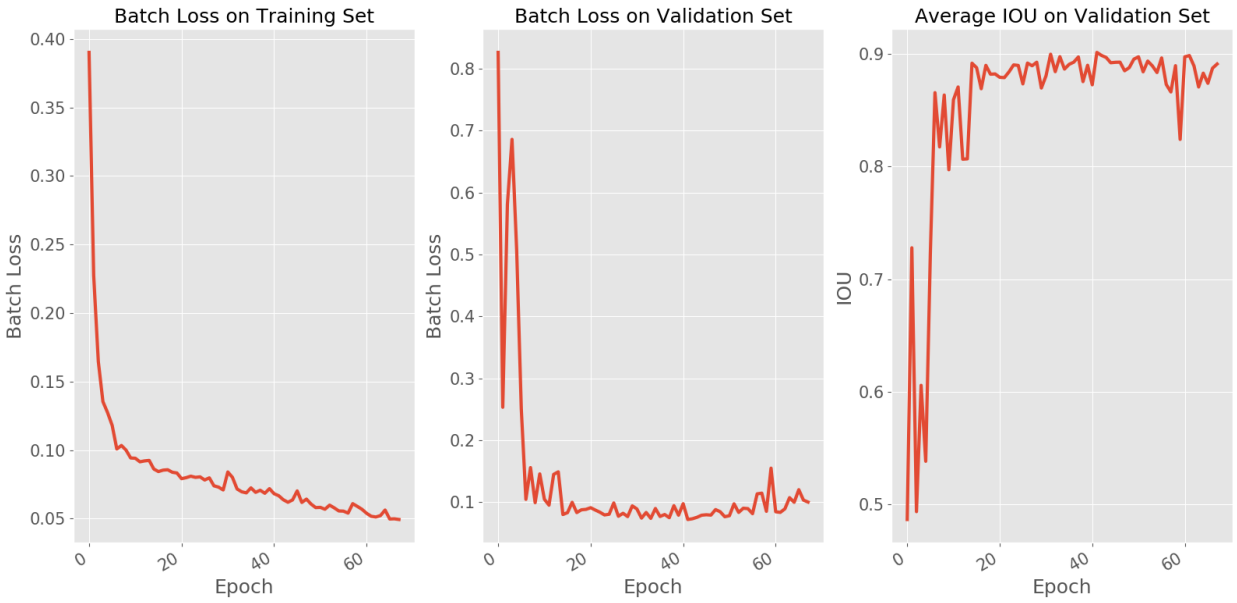

Figure 4-6. The loss on the training set (left), and validation set (middle). The average Intersect-Over-Union on validation set.

Since the purpose of the segmentation is not precisely segment the lung but remove the other unrelated features for better classification, the lung of the 1000 training images are roughly labeled by myself. The total images are divided into 800 images for training and 200 images for validation. The final IoU is around 0.9.

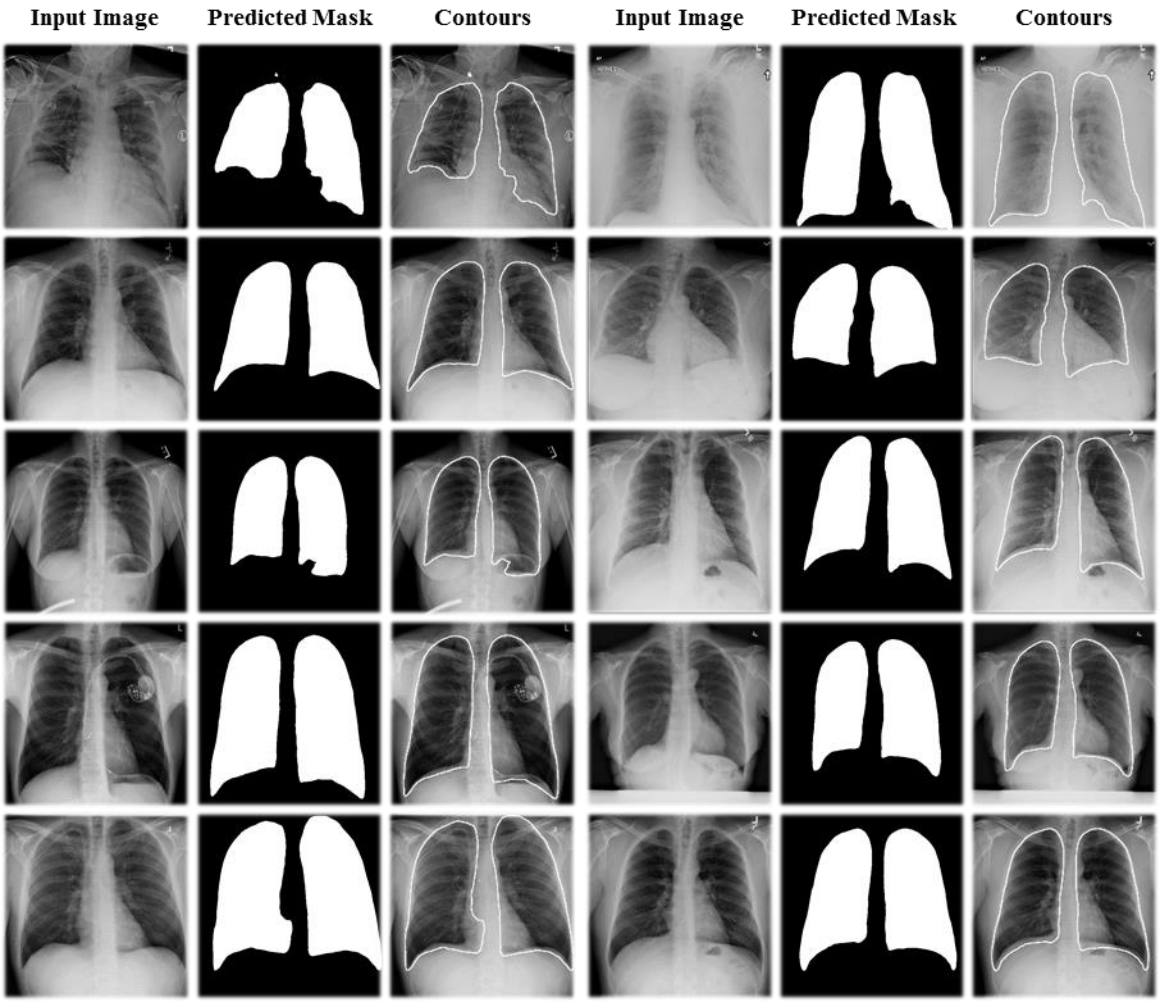

Figure 4-7. U-Net results on RSNA pneumonia detection testing dataset. The images are in dimension 512x512.

4.4. Fetus Brain Segmentation

Figure 4-8 shows the segmented areas of a fetal brain with the gestational age of 31 weeks in the test dataset. The green contours represent the manually segmented ground truth. The red contours represent the predicted segmentation by the customized U-Net model, the customized U-Net model with GAN loss, and FSL's BET program, respectively. The overlapped contours of the ground truth and predicted segmentation are shown in yellow. The results show that deep learning based segmentation has high resemblance in shape and size to the manual segmentation. The quantitative measures for performance evaluation are shown in the tables. Other than the Dice coefficient, the

IOU score provides an intuitive way to illustrate the resemblance between the ground truth and predicted segments. Both techniques have high sensitivity and specificity scores.

4.4.1. SAG dataset: Customized U-Net

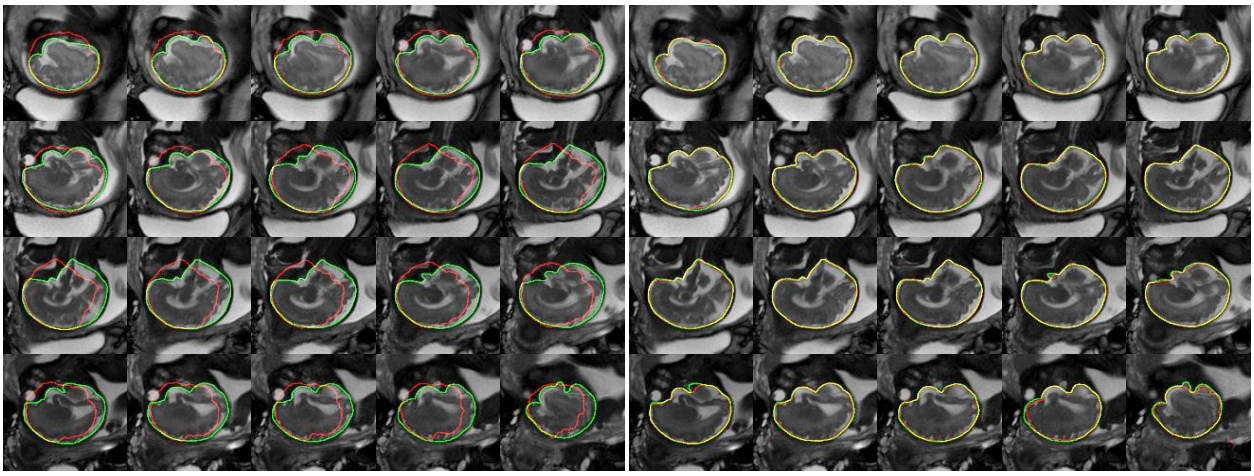

Figure 4-8. The BET segmentation results on SAG dataset (left). The U-Net segmentation results on SAG dataset (right). The green contour represents the ground truth label, and the red contour represents the contour predicted by U-Net model. The yellow contour is the overlay of green contour and red contour.

Table 4-1. The comparison of evaluation results between BET and U-Net on SAG dataset. It is shown that U-Net achieves 12.85% higher average Dice and 21.65% higher average IoU than BET method

Method	IOU	Dice	Sensitivity	Specificity
BET	74.94%	85.42%	82.98%	98.94%
U-Net	96.59%	98.27%	97.36%	99.92%

4.4.2. COR dataset: Customized U-Net

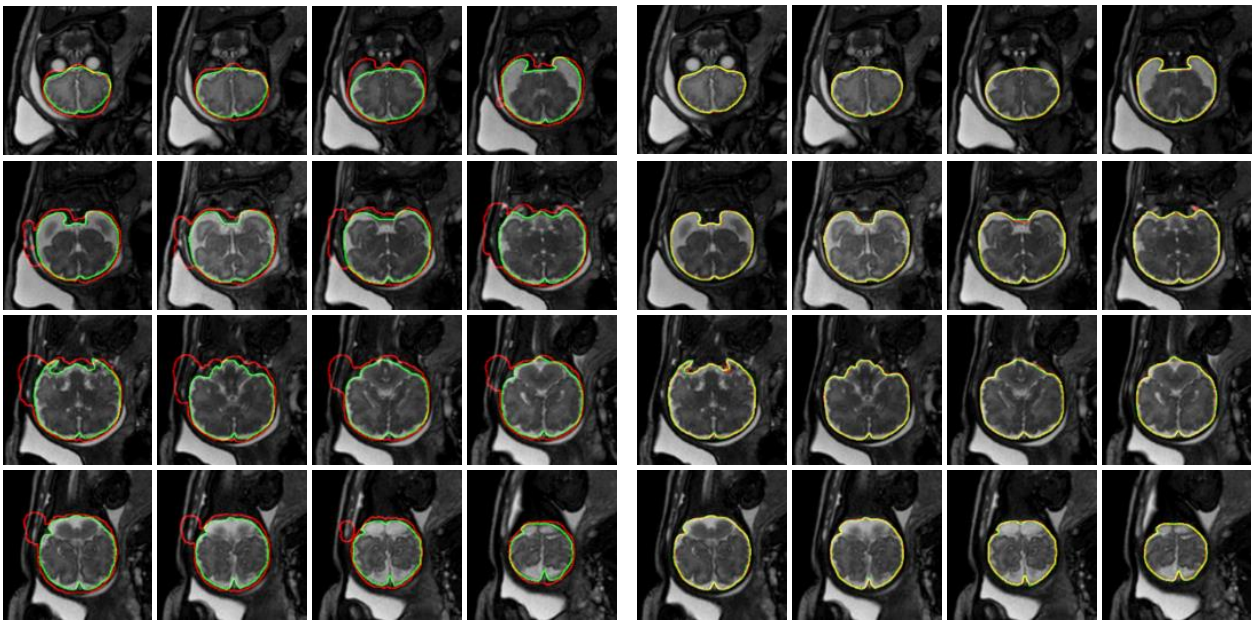

Figure 4-9. The BET segmentation results on COR dataset (left). The U-Net segmentation results on COR dataset (right). The green contour represents the ground truth label, and the red contour represents the contour predicted by U-Net model. The yellow contour is the overlay of green contour and red contour.

Table 4-2. The comparison of evaluation results between BET and U-Net on COR dataset. It is shown that U-Net achieves 9.88% higher average Dice and 17.06% higher average IoU than BET method

Method	IOU	Dice	Sensitivity	Specificity
BET	65.47%	78.81%	95.98%	97.02%
U-Net	94.63%	97.05%	99.35%	99.67%

4.4.3. TRA dataset: Customized U-Net

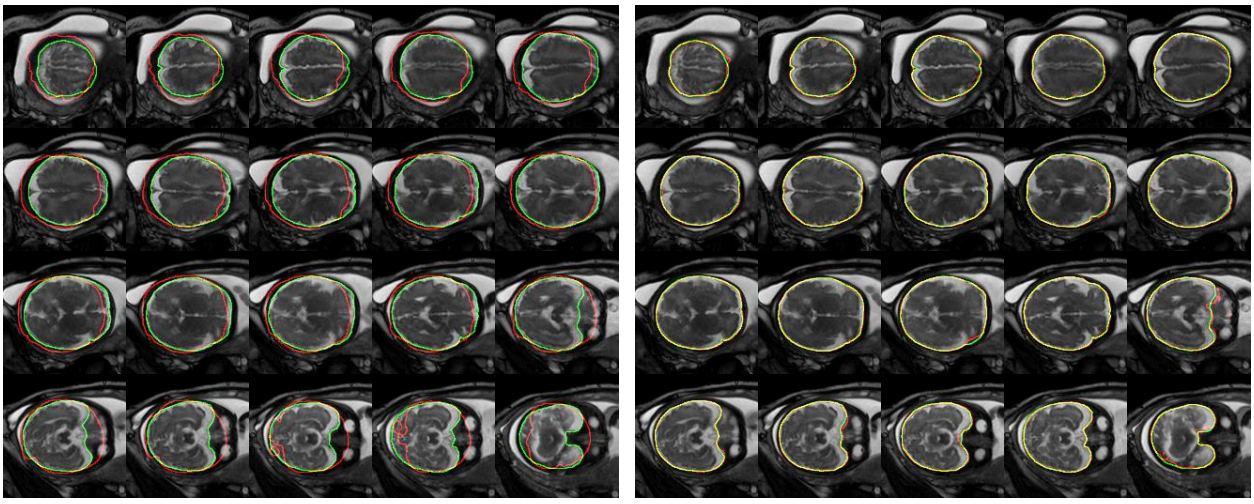

Figure 4-10. The BET segmentation results (left) and the U-Net segmentation results (right) on TRA dataset. The green contour represents the ground truth label, and the red contour represents the contour predicted by U-Net model. The yellow contour is the overlay of green contour and red contour.

Table 4-3. The comparison of evaluation results between BET and U-Net on TRA dataset. It is shown that U-Net achieves 9.56% higher average Dice and 16.59% higher average IoU than BET method

Method	IOU	Dice	Sensitivity	Specificity
BET	80.60%	89.12%	93.63%	98.31%
U-Net	97.19%	98.68%	98.41%	99.89%

4.4.4. SAG dataset: Customized FPN

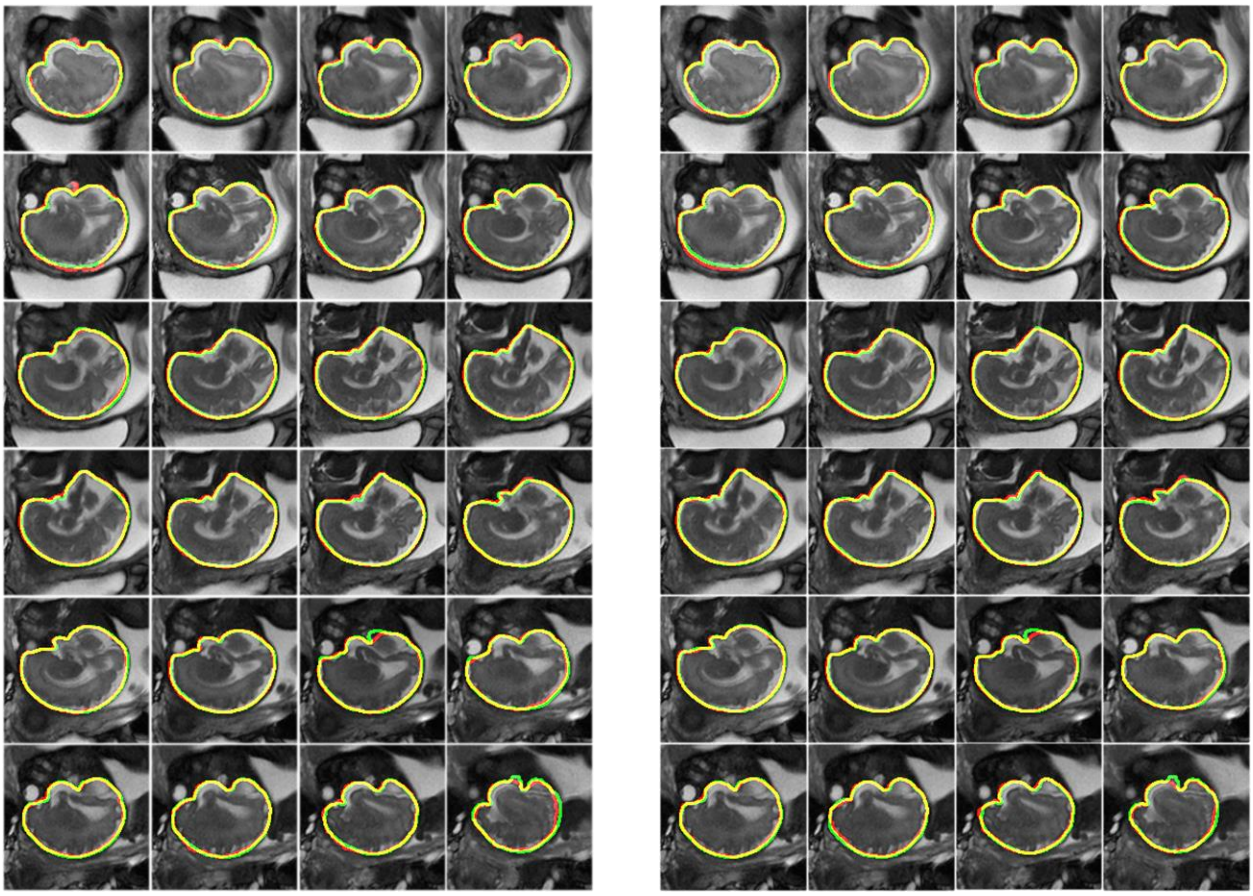

Figure 4-11. The FPN segmentation results (left) and the U-Net segmentation results (right) on SAG dataset.

Table 4-4. The comparison of evaluation results between FPN and U-Net on SAG dataset. It is shown that U-Net achieves 0.21% higher average Dice and 0.40% higher average IoU than FPN method

Method	IOU	Dice	Sensitivity	Specificity
FPN	96.29%	98.05%	97.21%	99.90%
U-Net	96.69%	98.26%	97.64%	99.90%

4.4.5. COR dataset: Customized FPN

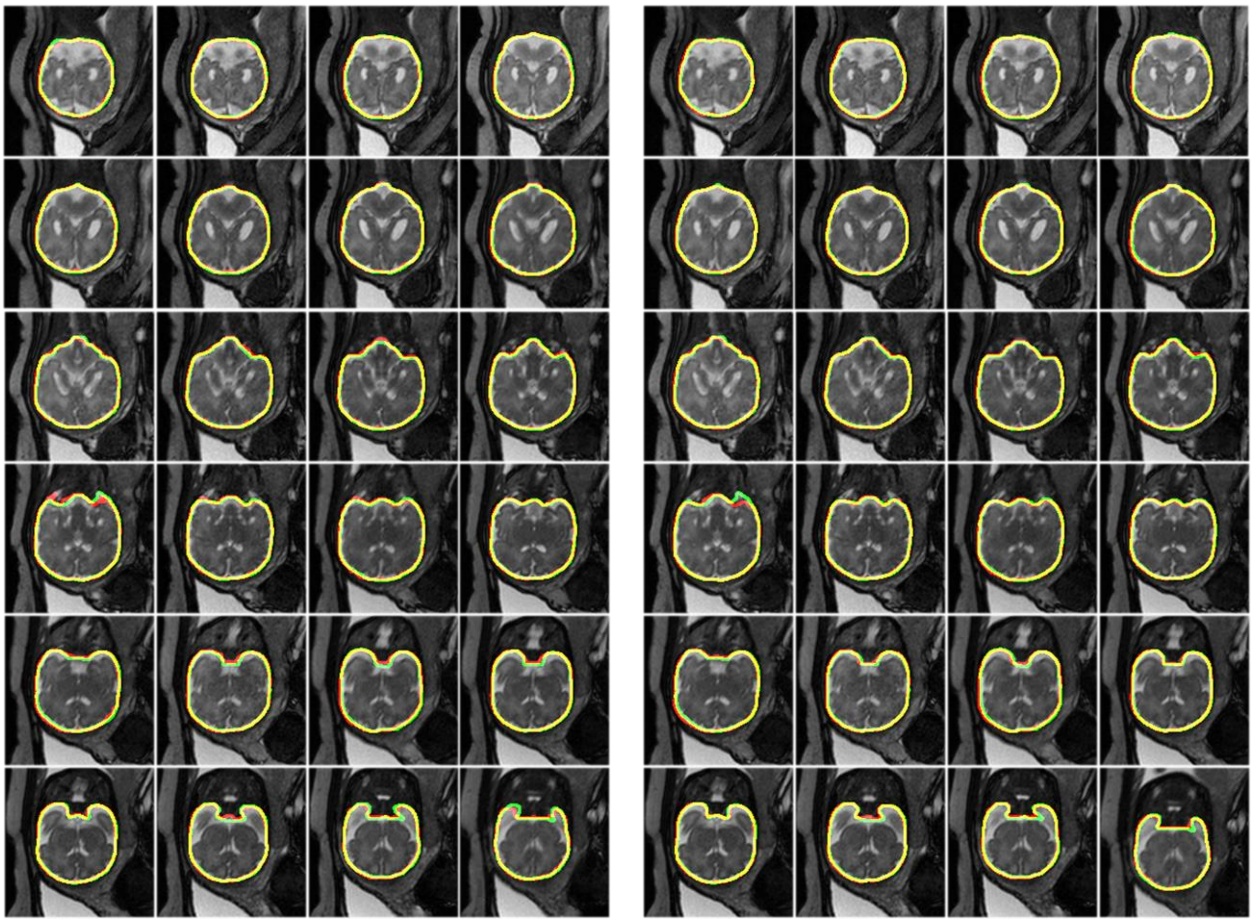

Figure 4-12. The FPN segmentation results (left) and the U-Net segmentation results (right) on COR dataset.

Table 4-5. The comparison of evaluation results between FPN and U-Net on COR dataset. It is shown that U-Net achieves 0.38% higher average Dice and 0.72% higher average IoU than FPN method

Method	IOU	Dice	Sensitivity	Specificity
FPN	96.73%	98.27%	97.58%	99.92%
U-Net	97.45%	98.65%	98.35%	99.92%

4.4.6. TRA dataset: Customized FPN

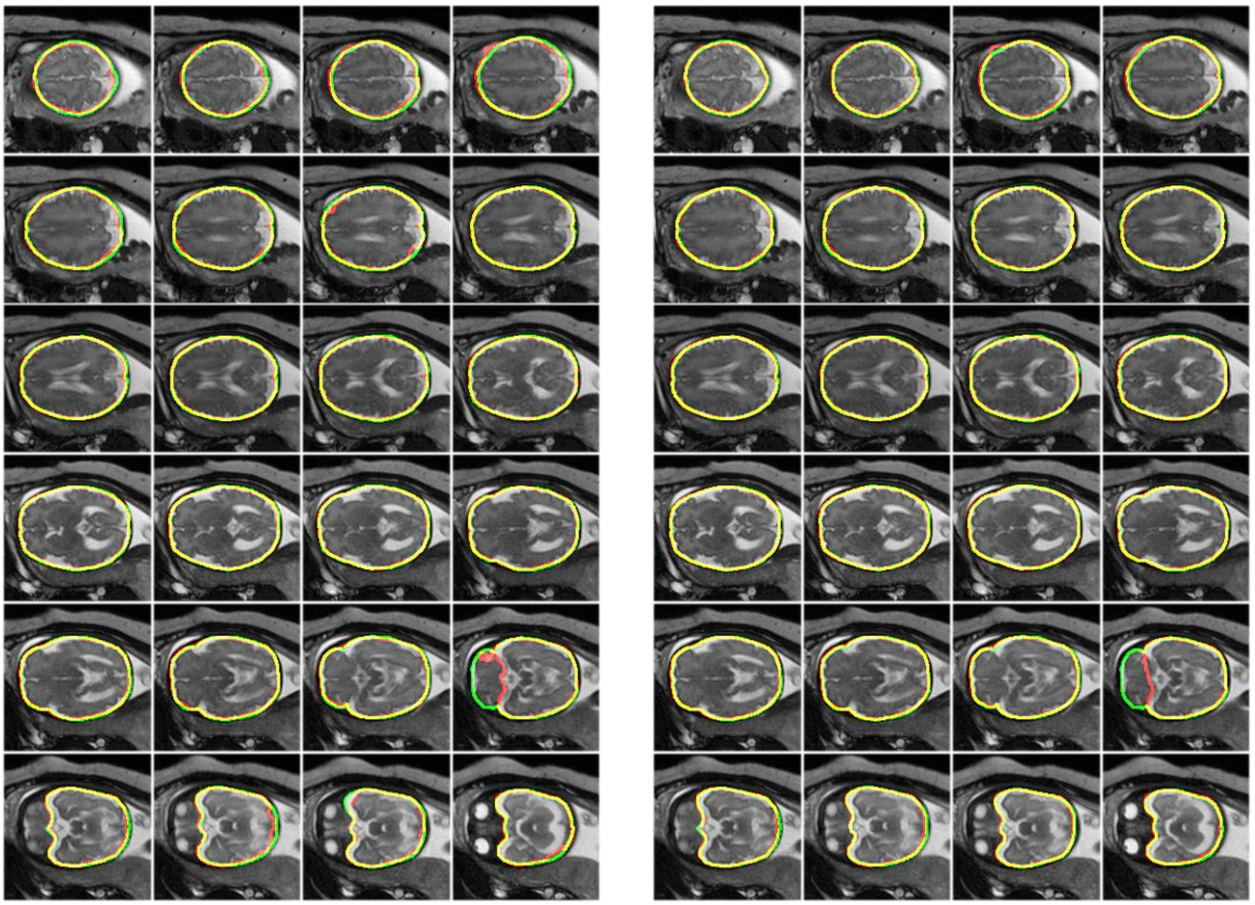

Figure 4-13. The FPN segmentation results (left) and the U-Net segmentation results (right) on TRA dataset.

Table 4-6. The comparison of evaluation results between FPN and U-Net on TRA dataset. It is shown that U-Net achieves 0.77% higher average Dice and 1.4% higher average IoU than FPN method

Method	IOU	Dice	Sensitivity	Specificity
FPN	95.46%	97.60%	97.58%	99.87%
U-Net	96.86%	98.37%	96.51%	99.89%

The predicted results from the FPN and U-Net show that the FPN is more likely to overfit the images and generate the high frequency components. However, the U-Net is more likely to generate smooth contour which represents the low frequency components.

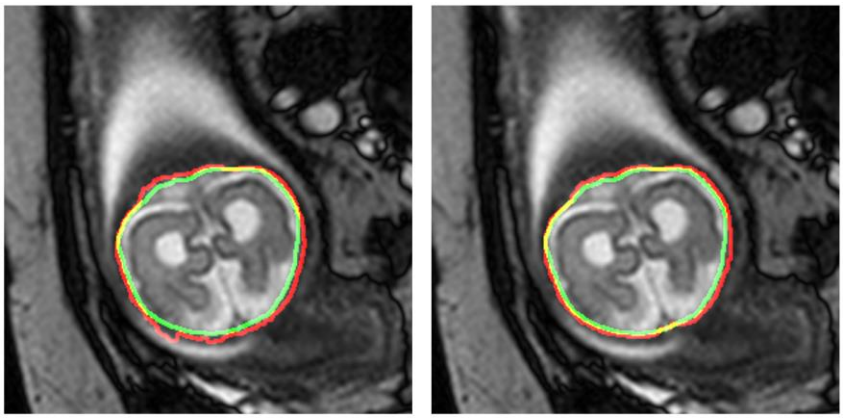

Figure 4-14. One sample for the FPN segmentation results (left) and one sample for the U-Net segmentation results (right) on TRA dataset.

4.4.7. SAG dataset: Customized U-Net + GAN Loss

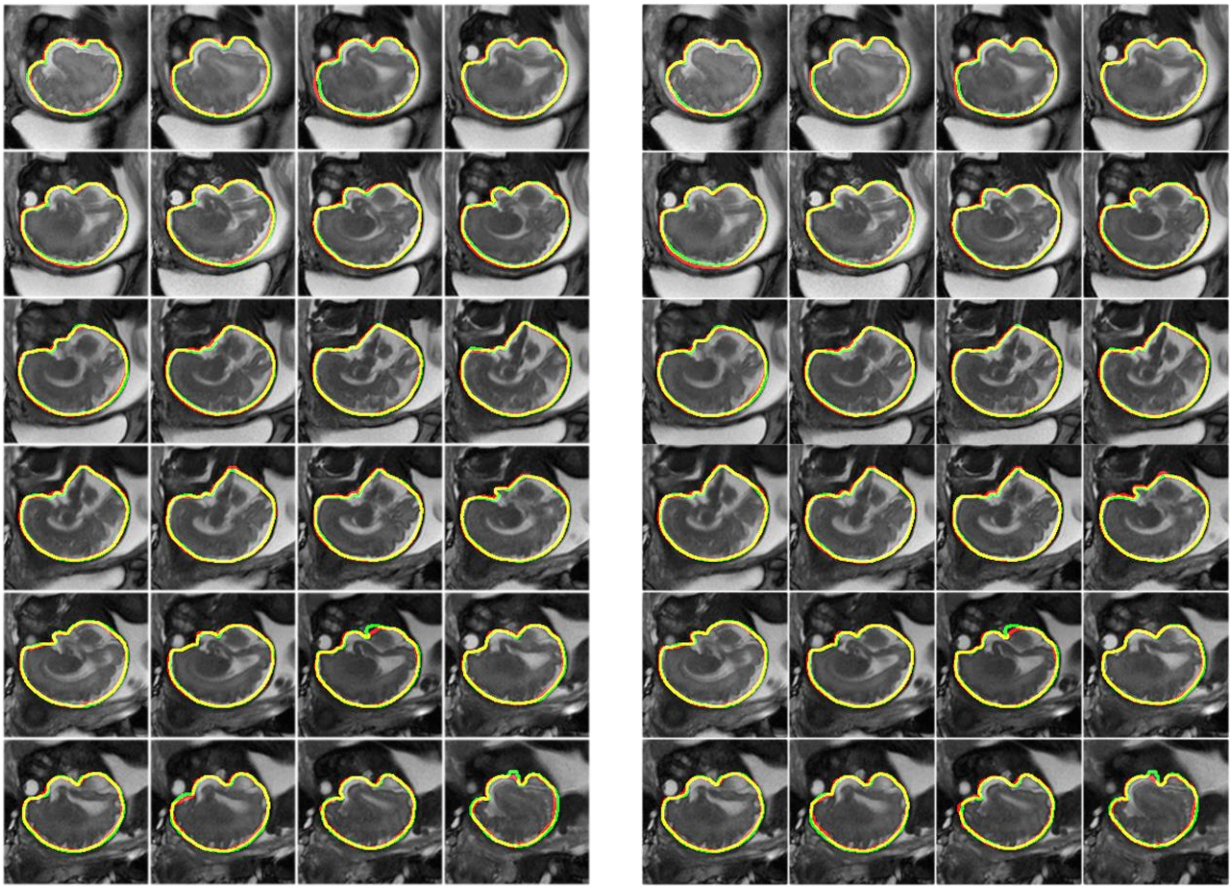

Figure 4-15. The U-Net with GAN loss results (left) and the U-Net segmentation results (right) on SAG dataset.

Table 4-7. The comparison of evaluation results between U-Net and U-Net with GAN Loss on SAG dataset. It is shown that U-Net achieves 0.08% higher average Dice and 0.17% higher average IoU than U-Net + GAN Loss method

Method	IOU	Dice	Sensitivity	Specificity
U-Net+GANLoss	96.52%	98.18%	97.50%	99.90%
U-Net	96.69%	98.26%	97.64%	99.90%

4.4.8. COR dataset: Customized U-Net + GAN Loss

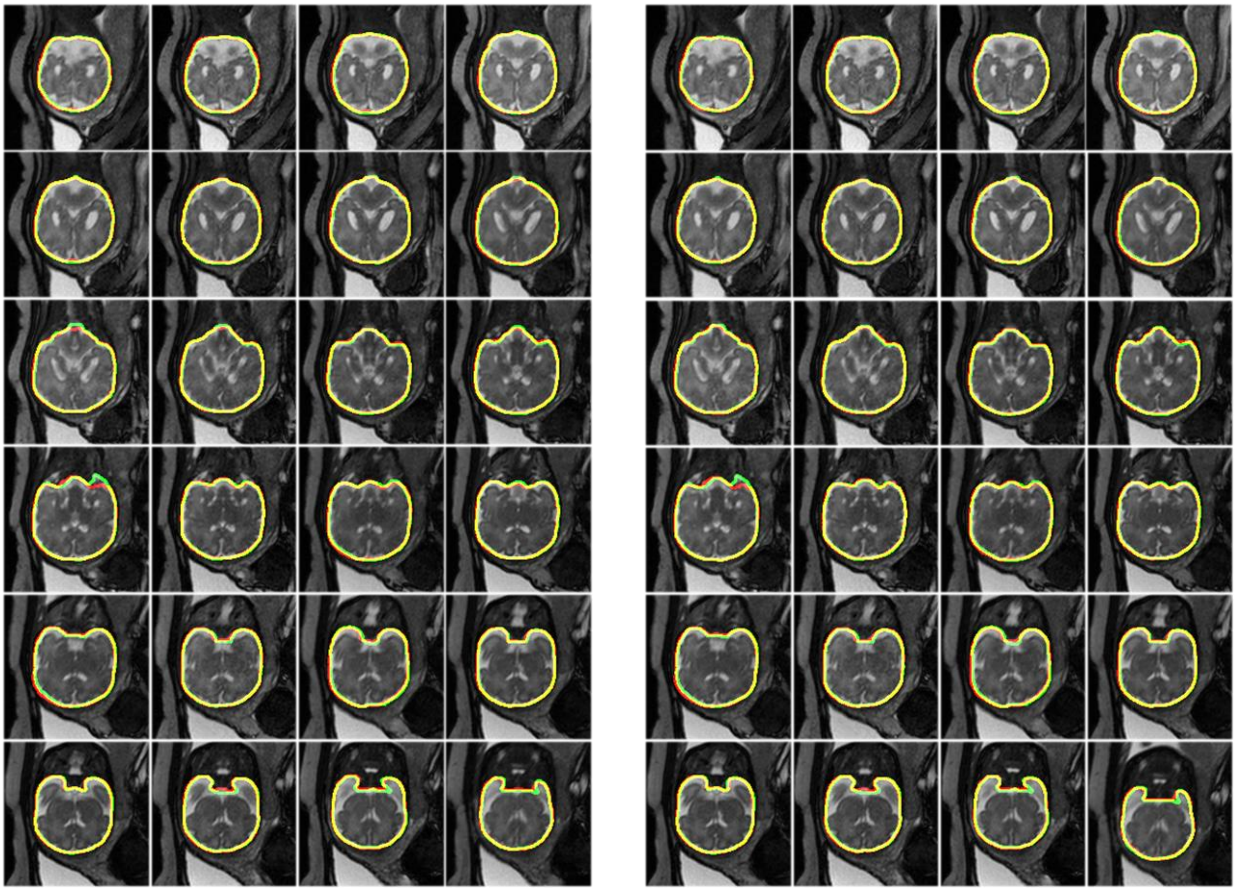

Figure 4-16. The U-Net with GAN loss results (left) and the U-Net segmentation results (right) on COR dataset.

Table 4-8. The comparison of evaluation results between U-Net and U-Net with GAN Loss on COR dataset. It is shown that U-Net achieves 0.07% higher average Dice and 0.15% higher average IoU than U-Net + GAN Loss method

Method	IOU	Dice	Sensitivity	Specificity
U-Net+GANLoss	97.30%	98.58%	98.16%	99.93%
U-Net	97.45%	98.65%	98.35%	99.92%

4.4.9. TRA dataset: Customized U-Net + GAN Loss

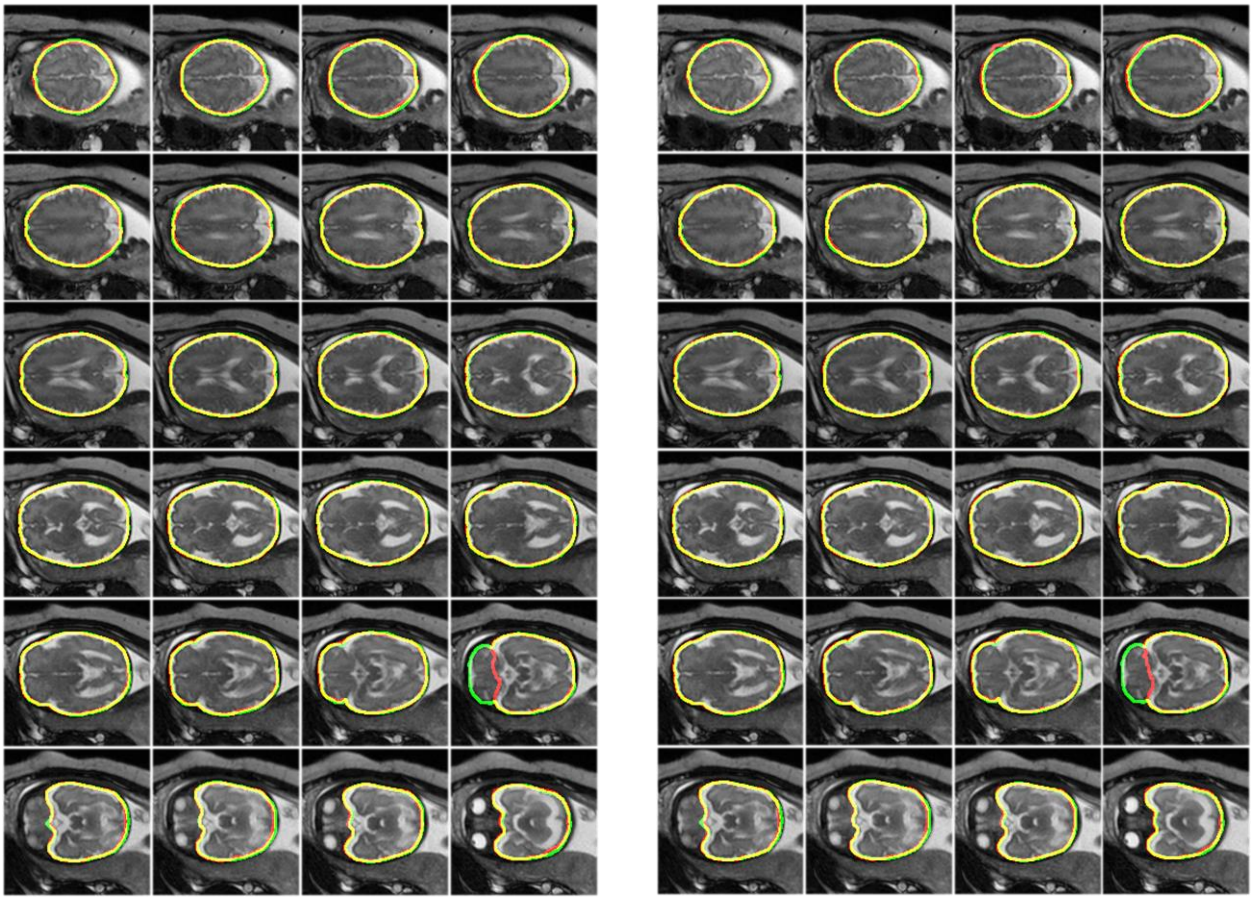

Figure 4-17. The U-Net with GAN loss results (left) and the U-Net segmentation results (right) on TRA dataset.

Table 4-9. The comparison of evaluation results between U-Net and U-Net with GAN Loss on TRA dataset. It is shown that U-Net achieves 0.49% higher average Dice and 0.86% higher average IoU than U-Net + GAN Loss method

Method	IOU	Dice	Sensitivity	Specificity
U-Net+GANLoss	96.00%	97.88%	96.22%	99.97%
U-Net	96.86%	98.37%	96.51%	99.89%

4.4.10. SAG dataset: customized U-Net + Focal Loss

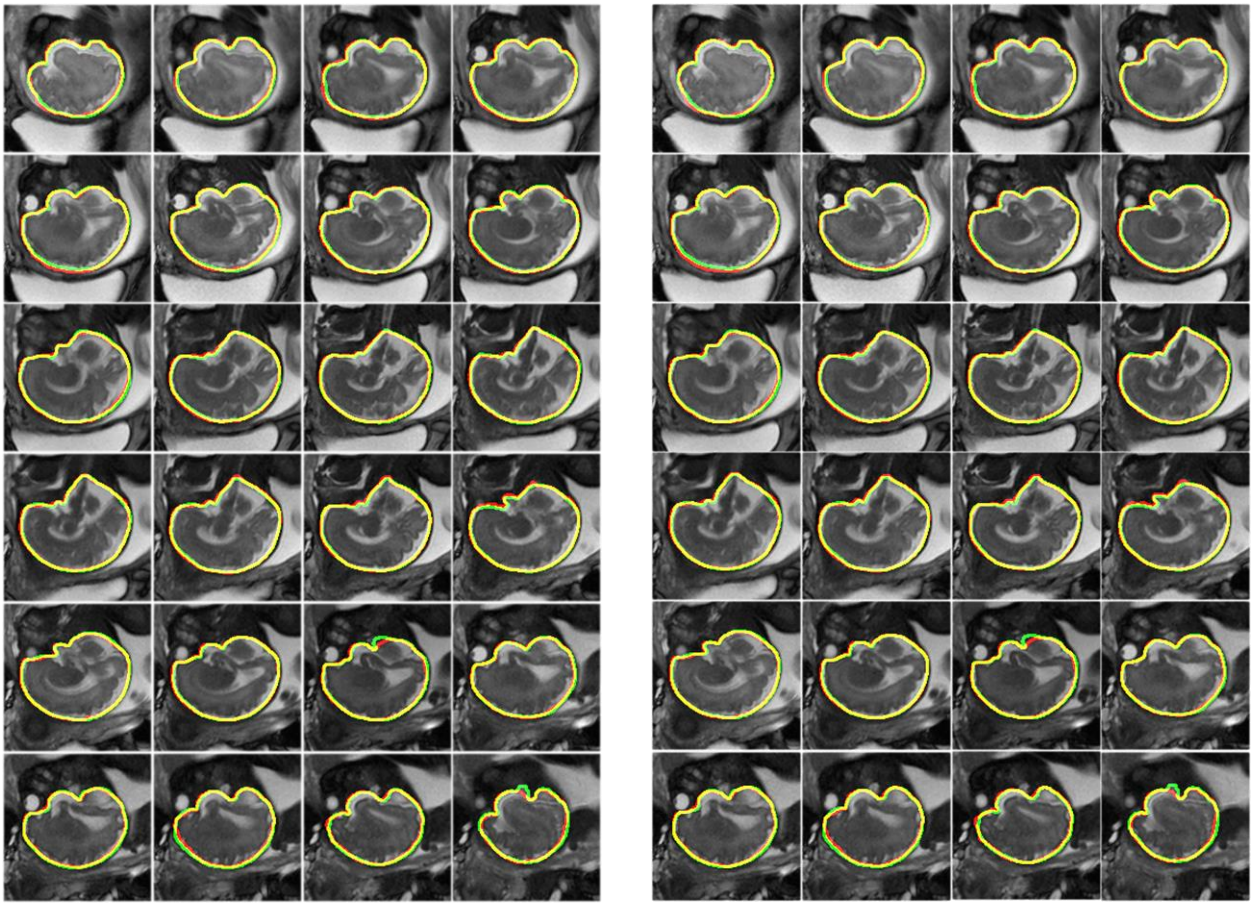

Figure 4-18. The U-Net with Focal Loss results (left) and the U-Net segmentation results (right) on SAG dataset.

Table 4-10. The comparison of evaluation results between U-Net and U-Net with Focal Loss on SAG dataset. It is shown that U-Net achieves 0.05% higher average Dice and 0.10% higher average IoU than U-Net with Focal Loss method

Method	IOU	Dice	Sensitivity	Specificity
U-Net + Focal Loss	96.59%	98.21%	97.38%	99.91%
U-Net	96.69%	98.26%	97.64%	99.90%

4.4.11. COR dataset: customized U-Net + Focal Loss

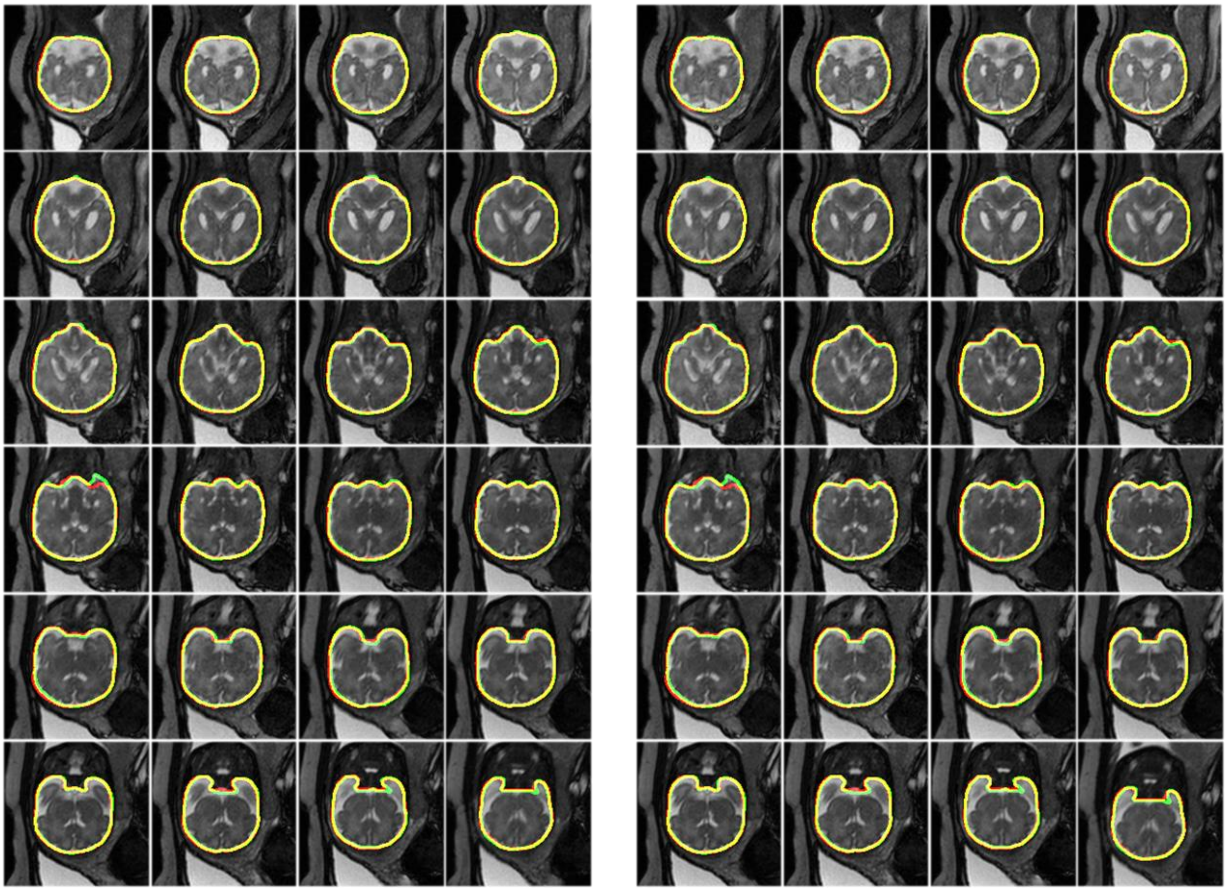

Figure 4-19. The U-Net with Focal Loss results (left) and the U-Net segmentation results (right) on COR dataset.

Table 4-11. The comparison of evaluation results between U-Net and U-Net with Focal Loss on COR dataset. It is shown that U-Net achieves 0.01% lower average Dice and 0.18% lower average IoU than U-Net with Focal Loss method

Method	IOU	Dice	Sensitivity	Specificity
U-Net + Focal Loss	97.63%	98.64%	98.26%	99.92%
U-Net	97.45%	98.65%	98.35%	99.92%

4.4.12. TRA dataset: customized U-Net + Focal Loss

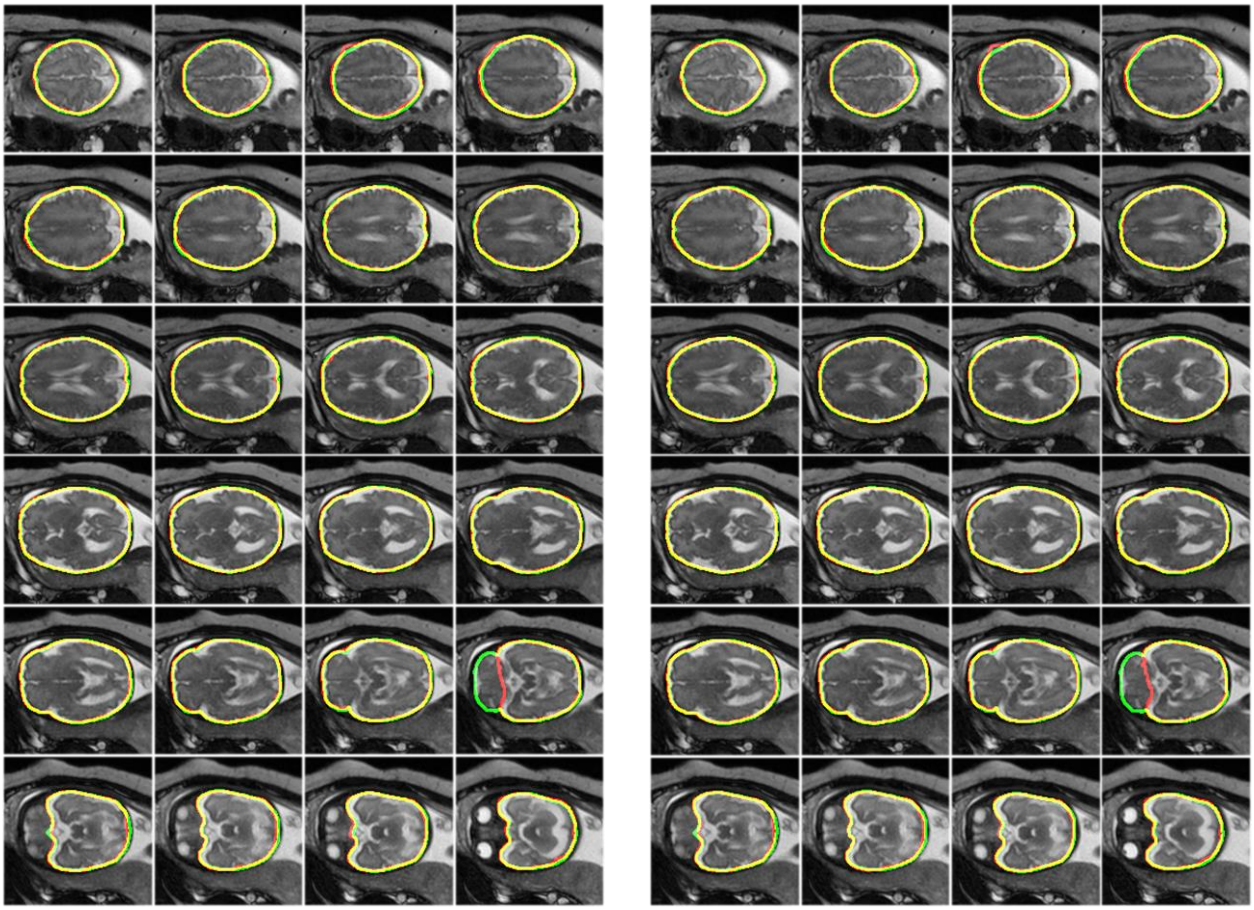

Figure 4-20. The U-Net with Focal Loss results (left) and the U-Net segmentation results (right) on TRA dataset.

Table 4-12. The comparison of evaluation results between U-Net and U-Net with Focal Loss on TRA dataset. It is shown that U-Net achieves 0.60% lower average Dice and 1.31% lower average IoU than U-Net with Focal Loss method

Method	IOU	Dice	Sensitivity	Specificity
U-Net + Focal Loss	97.63%	98.64%	96.65%	99.96%
U-Net	96.32%	98.04%	96.51%	99.89%

4.4.13. Statistical Results

Table 4-13. The statistical results for all methods on coronal dataset.

	Patient #1		Patient #2		Patient #3	
Method	IOU	Dice	IOU	Dice	IOU	Dice
BET	72.15%	82.75%	89.26%	93.92%	82.81%	89.83%
U-Net	96.20%	98.03%	97.45%	98.65%	96.75%	98.29%
U-Net + GAN Loss	96.03%	97.93%	97.30%	98.58%	96.65%	98.24%
FPN	95.63%	97.71%	96.73%	98.27%	96.25%	98.03%
U-Net+Focal Loss	96.22%	98.03%	97.43%	98.64%	96.52%	98.17%
FPN+Focal Loss	95.23%	97.49%	96.62%	98.20%	96.14%	97.96%

Table 4-14. The statistical results for all methods on sagittal dataset.

	Patient #1		Patient #2		Patient #3	
Method	IOU	Dice	IOU	Dice	IOU	Dice
BET	70.52%	81.61%	86.41%	92.77%	81.56%	89.28%
U-Net	95.76%	97.74%	96.69%	98.26%	94.32%	96.92%
U-Net + GAN Loss	95.49%	97.59%	96.52%	98.18%	96.14%	97.96%
FPN	95.09%	97.38%	96.29%	98.05%	95.16%	97.45%
U-Net+Focal Loss	95.75%	97.74%	96.59%	98.21%	94.32%	96.88%
FPN+Focal Loss	94.63%	97.11%	95.99%	97.87%	94.63%	97.14%

Table 4-15. The statistical results for all methods on transverse dataset.

	Patient #1		Patient #2		Patient #3	
Method	IOU	Dice	IOU	Dice	IOU	Dice
BET	84.48%	90.62%	65.10%	78.69%	84.93%	91.53%
U-Net	95.70%	97.66%	96.24%	98.00%	96.86%	98.37%
U-Net + GAN Loss	95.64%	97.66%	96.00%	97.88%	96.53%	98.20%
FPN	93.84%	96.52%	94.77%	97.18%	95.45%	97.60%
U-Net+Focal Loss	95.46%	97.52%	96.32%	98.04%	96.95%	98.42%
FPN+Focal Loss	93.67%	96.39%	94.63%	97.08%	94.84%	97.25%

CHAPTER 5. DISCUSSION

In this chapter, we will discuss the problems we encountered during the experiment and provide more thoroughly error analysis.

5.1. Customized U-Net

Compare with the original U-Net, the batch normalization is added after each convolution layers to boost the convergence of the network. We found it works very good even a large learning rate is used. Instead of using the popular optimizer Adam, we found that the SGD optimizer can achieve much faster convergence speed and higher segmentation IoUs at the end. Also, the using of Adam increases the computation which significantly delays the training progress. The author of U-Net did not use paddings during the downsampling which makes the dimension of the feature inconsistency during the upsampling, so the feature crop is needed during the concatenation. We fixed this by adding the paddings and make the features in downsampling and upsampling symmetric and removed the feature crop procedure.

Because of the binary cross-entropy loss function for U-Net, the prediction will be a probability mask in the range of 0 and 1. We noticed that sometimes there are tiny holes in the mask or tiny isolated pixels outside the masks. This contributes little to the overall IoUs but looks very weird from the shape. Because of the property of binary cross-entropy loss function, this problem cannot be solved even training more epochs. We tried to replace the binary cross-entropy function with the focal loss [58] to suppress the convergence speed of negative samples and speed up the convergence of the positive samples, but this does not solve the problem. Finally, end up using the traditional image processing algorithm to find the contours from the binary mask and the problem was solved successfully. However, this only works for fetus brain dataset because there is only one object for each image. Our proposed pipeline for solving this problem is shown in the following.

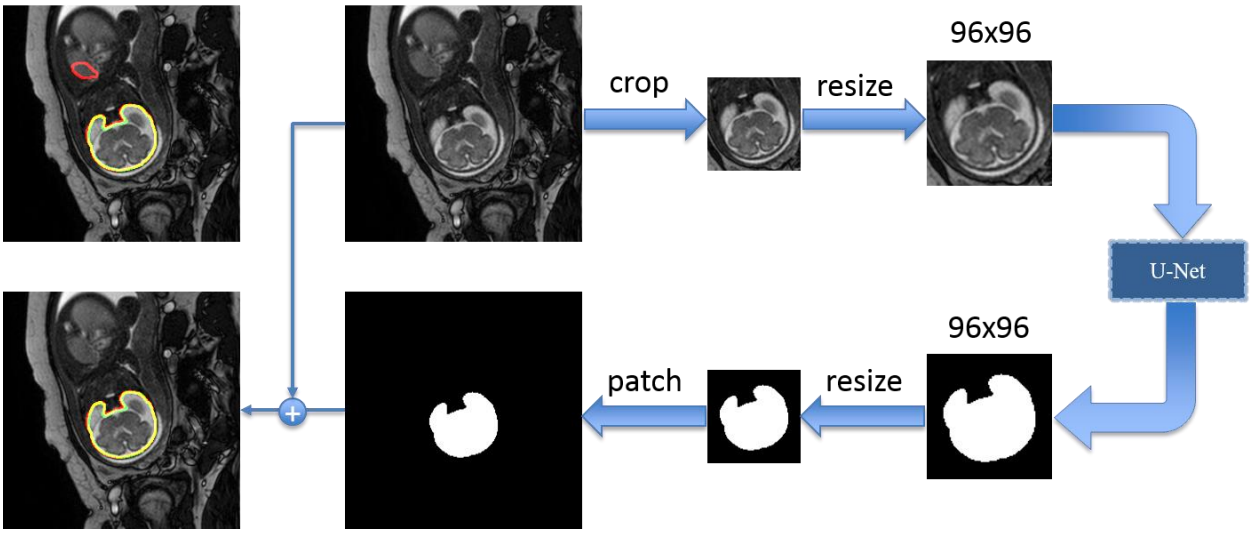

Figure 5-1. The pipeline we proposed to solve the isolated predictions

5.2. U-Net with GAN Loss

To increase the segmentation accuracy, we adopted the adversarial training inspired by the generative adversarial networks. We used the pre-trained U-Net weights as the generator and added the extra loss from the discriminator, which is also an encoder. In the beginning, we found the discriminator always converges faster than the generator, finally, we found the reason is we feed the binary masks to the discriminator as the negative and feed the probability masks to the discriminator as the positive, which makes the discriminator distinguish the fake and real images easily. So, we end up converting the probability mask to binary mask when training the discriminator.

The original GAN trains the generator and discriminator alternatively, updating each of them on one batch data. However, since our image is in high dimension (512x512), only a batch of data (8 images) cannot represent the whole data distribution in high dimension. So it is difficult to make the distribution of the fake data overlaps with the distribution of the real data, which makes it difficult to train the network. So, instead of one batch alternative training, we train the discriminator on all the data and then train the generator on all the data iteratively, this significantly increased the convergence speed and ended up having a higher final IoUs.

5.3. Difficulties of Pneumonia Detection

As discussed in chapter 3, the pneumonia dataset contains 25000 2D single channel CT images along with corresponding labels. There are three categories in the dataset, ‘normal’, ‘pneumonia’, and ‘not normal’. The ‘not normal’ category could be cancer or something else except pneumonia. So, only the ‘pneumonia’ category has the ground truth bounding boxes of pneumonia, which means the Faster R-CNN model cannot utilize the ‘normal’ and ‘not normal’ categories. Thus, we dropped the ‘normal’ and ‘not normal’ data and only use ‘pneumonia’ data. We end up having ~5000 images, which significantly reduced the training samples for the Faster R-CNN model. Moreover, because of the extreme similarity between the features of ‘not normal’ category and the ‘pneumonia’ category, the Faster R-CNN have difficulties distinguish them and cased many false detections among these two categories. I also tried to implement another deep network such as ResNet or DenseNet to classify the ‘pneumonia’ and ‘not normal’ categories, but could not achieve promising results. So our model can localize pneumonia accurately if there are no ‘not normal’ samples in the testing set.

There are also some failure cases of detecting pneumonia. Since the dataset contains healthy subjects, pneumonia subjects and abnormal subjects three categories, and due to the similarity of the pneumonia subjects and abnormal subjects (maybe cancer or other diseases that cause the feature of the lung looks similar as the pneumonia cases), the Faster R-CNN have difficulty distinguish pneumonia and abnormal. So it will mark all of them as pneumonia, which caused many false positive predictions.

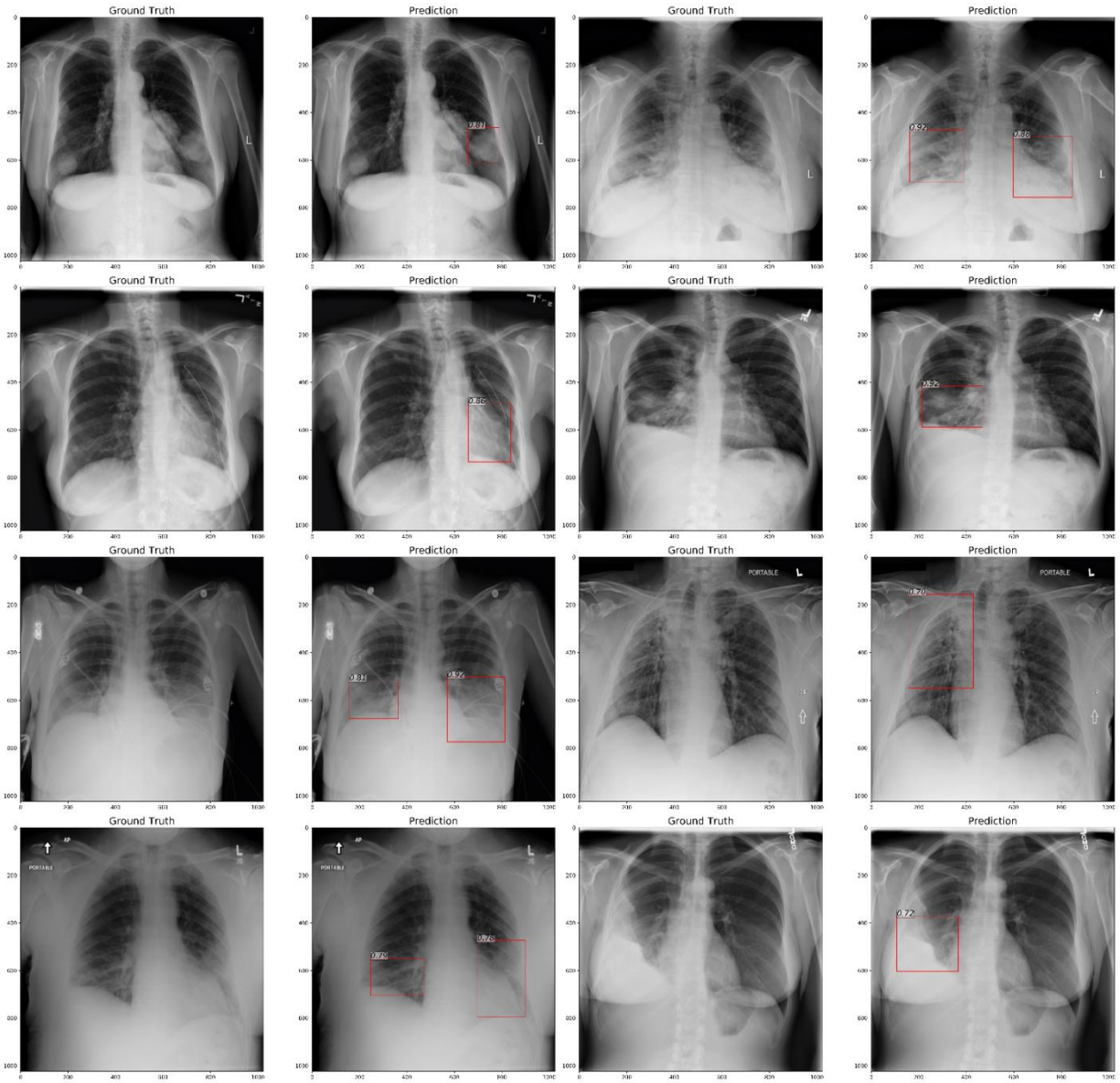

Figure 5-2. False Positive. Faster R-CNN predicts many abnormal lung as pneumonia.

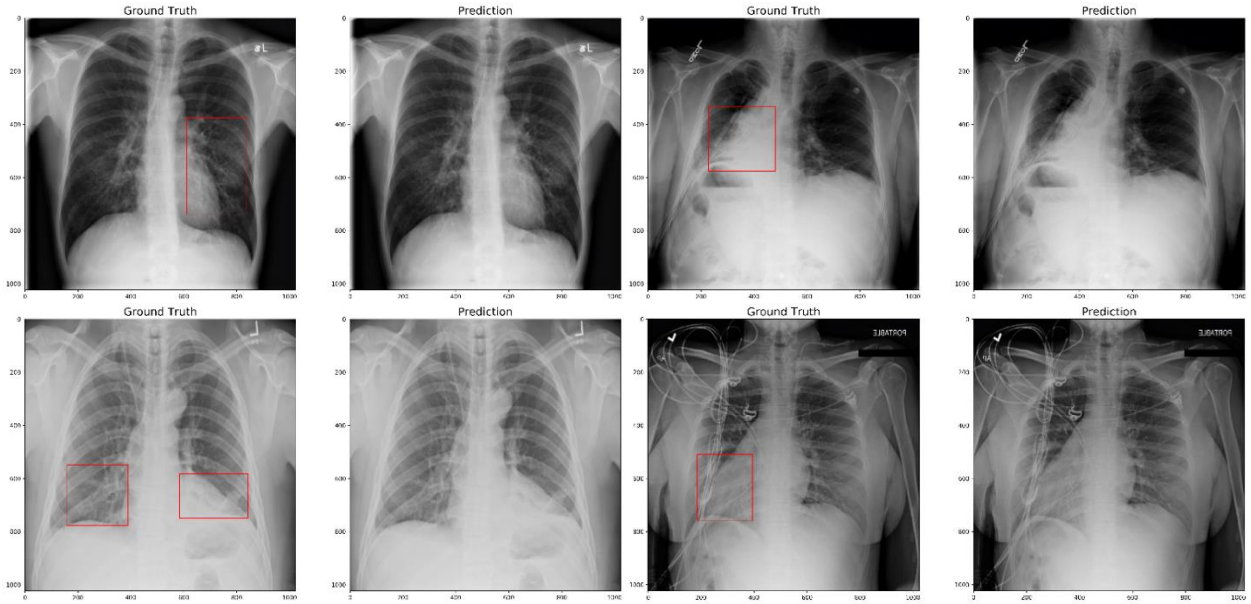

Figure 5-3. False Negative. For some pneumonia subjects, the Faster R-CNN could not detect any pneumonia area.

CHAPTER 6. CONCLUSION

In this chapter, we will review our results and provide some concluding remarks. We start by reviewing the current methods for object detection and segmentation. Then we will discuss the implementation of these methods and finally to explore the potential improvements.

We started the thesis with an overview of the current development and challenges in computer vision. In chapter 2, we illustrated the algorithms of some traditional feature extraction algorithm in computer vision such as the histogram of oriented gradient, and the details of the algorithm in machine learning, such as logistic regression and support vector machine for classification. HOG and SVM are classic combinations to extract the features and classify these features before the neural network come out. Then we reviewed the mechanism of neural networks and explained why the single layer perceptron could not solve the non-linear problem. We illustrated how the backpropagation algorithm works and how to tune the hyper parameters in the neural networks. Then we get into the core part of computer vision, the convolutional neural networks. We demonstrated how convolutional neural networks work and why it is so useful for extracting features compare with the fully connected networks. We explained the transposed convolution for feature upsampling and reconstruction that we used in our customized U-Net model and FPN model. Finally, we reviewed the current state-of-the-art models for object detection, localization and segmentation. We gave an overview of VGG, ResNet, FPN, U-Net and GAN and discussed the mathematics behind the algorithm.

In chapter 3, we discussed the implementation of object detection and segmentation thoroughly. We started from the environment setup and the introduction of the dataset for the experiments, nucleus dataset, pneumonia dataset and fetus brain dataset. We tested our customized U-Net model on the nucleus dataset to segment the nuclei out from the background. We used the pneumonia dataset for object detection, the U-Net to segment the lung out and used the Faster R-CNN to localize pneumonia. Finally, we proposed an automatic fetus brain segmentation approach using the combination of our customized U-Net and the recently emerged power model GAN. We reviewed the detailed pipeline of Faster R-CNN. It consists of three stages, the feature extraction stage, region proposal stage, and the RoIHead stage. We discussed the algorithms for each stage

and how to implement them. We illustrated the architecture of our customized U-Net model and how to combine it with the GAN for training. Then we looked into the evaluation metrics for object detection and segmentation.

In chapter 4, we presented the final results that we achieved base on our implementation and compared our segmented results with the popular method BET for fetus brain segmentation. Our deep learning based model achieves much better results than BET. Our results will be further used in fetal brain reconstruction. Finally, we wrapped them up by error analysis and future potential improvement discussion. We tested the dataset on many different deep learning models with different loss functions; they achieved quite a similar accuracy. Sometimes their performance depends on the MRI scan type.

In summary, this thesis presented the customized model based on the current state-of-the-art deep learning models for object detection and segmentation and achieved excellent results on the application of the biomedical image.

BIBLIOGRAPHY

[1] D. Liu, Y. Xiong, K. Pulli, and L. Shapiro, "Estimating image segmentation difficulty," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[3] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017.

[5] C. Smailis, "Mask RCNN," *ICCV*, 2017.

[6] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, 2005.

[7] L. J. Davis and K. P. Offord, "Logistic regression," *Journal of Personality Assessment*. 1997.

[8] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica (Ljubljana)*. 2007.

[9] D. Meyer, "Support Vector Machines," *R News*, 2001.

[10] R. G. Brereton and G. R. Lloyd, "Support Vector Machines for classification and regression," *Analyst*. 2010.

[11] A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, *Advances in Large Margin Classifiers*. 2000.

[12] L. Jäncke, "The plastic human brain," *Restorative Neurology and Neuroscience*. 2009.

[13] A. Bhaya, "Neural networks," in *Decision Sciences: Theory and Practice*, 2016.

[14] Y. Freund and R. E. Schapire, "Large margin classification using the perceptron algorithm," *Mach. Learn.*, 1999.

[15] M. Minsky and S. Papert, "Perceptrons," in *Perceptrons: An Introduction to Computational Geometry*, 1969.

[16] Kunihiko Fukushima, “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position Kunihiko,” *Biol. Cybern.*, 1980.

[17] D. Svozil, V. Kvasnička, and J. Pospíchal, “Introduction to multi-layer feed-forward neural networks,” in *Chemometrics and Intelligent Laboratory Systems*, 1997.

[18] H. Il Suk, “An Introduction to Neural Networks and Deep Learning,” *Deep Learning for Medical Image Analysis*, 2017.

[19] B. Karlik, “Performance analysis of various activation functions in generalized MLP architectures of neural networks,” *Int. J. Artif. Intell. Expert Syst.*, 2015.

[20] Z. Sankari and H. Adeli, “Probabilistic neural networks,” *J. Neurosci. Methods*, 2011.

[21] G. Zhao, Z. Zhang, J. Wang, and H. Guan, “Training Better CNNs Requires to Rethink ReLU,” *arXiv*, 2017.

[22] J. Kivinen and M. K. Warmuth, “Relative loss bounds for multidimensional regression problems,” *Mach. Learn.*, 2001.

[23] J. E. Shore and R. W. Johnson, “Properties of Cross-Entropy Minimization,” *IEEE Trans. Inf. Theory*, 1981.

[24] T. Liu, S. Fang, Y. Zhao, P. Wang, and J. Zhang, “Implementation of Training Convolutional Neural Networks,” *Found. Trends® Signal Process.*, 2015.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning Internal Representations by Error Propagation,” in *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, 2013.

[26] H. Leung and S. Haykin, “The complex backpropagation algorithm,” *IEEE Trans. Signal Process.*, 1991.

[27] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT 2010 - 19th International Conference on Computational Statistics, Keynote, Invited and Contributed Papers*, 2010.

[28] M. Li, T. Zhang, Y. Chen, and A. J. Smola, “Efficient mini-batch training for stochastic optimization,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’14*, 2014.

[29] “Artificial neural network.” [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network.

[30] G. E. Dahl, T. N. Sainath, and G. E. Hinton, “On the Importance of Initialization and Momentum in Deep Learning,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[31] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic gradient descent,” *ICLR Int. Conf. Learn. Represent.*, 2015.

[32] F. A. Spanhol, L. S. Oliveira, C. Petitjean, and L. Heutte, “Breast cancer histopathological image classification using Convolutional Neural Networks,” in *Proceedings of the International Joint Conference on Neural Networks*, 2016.

[33] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[34] C. N. dos Santos and M. Gatti, “Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts,” in *The 25th International Conference on Computational Linguistics*, 2014.

[35] A. Krizhevsky and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Neural Inf. Process. Syst.*, 2012.

[36] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, “Deconvolutional networks,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010.

[37] K. Simonyan and A. Zisserman, “VGG-16,” *arXiv Prepr.*, 2014.

[38] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, vol. pp. 770-77, 2015.

[39] J. S. Kaiming He, Xiangyu Zhang, Shaoqing Ren, “Deep Residual Learning for Image Recognition.”

[40] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[41] J. S. D. R. G. A. F. Redmon, “(YOLO) You Only Look Once,” *Cvpr*, 2016.

[42] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.

[43] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[44] Jonathan Hui, "Understanding Feature Pyramid Networks for object detection (FPN)." [Online]. Available: https://medium.com/@jonathan_hui/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c.

[45] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017.

[46] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015.

[47] Y. B. Ian J. Goodfellow, Jean Pouget-Abadie∗, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair†, Aaron Courville and Y. Y. Wang, "Generative Adversarial Nets," in *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2016.

[48] GoogleResearch, "TensorFlow: Large-scale machine learning on heterogeneous systems," *Google Res.*, 2015.

[49] N. Ketkar, "Introduction to PyTorch," in *Deep Learning with Python*, 2017.

[50] A. Makropoulos, S. J. Counsell, and D. Rueckert, "A review on automatic fetal and neonatal brain MRI segmentation," *NeuroImage*. 2018.

[51] K. Keraudren *et al.*, "Automated fetal brain segmentation from 2D MRI slices for motion correction," *Neuroimage*, 2014.

[52] S. Tourbier *et al.*, "Automated template-based brain localization and extraction for fetal brain MRI reconstruction," *Neuroimage*, 2017.

[53] N. Souly, C. Spampinato, and M. Shah, "Semi Supervised Semantic Segmentation Using Generative Adversarial Network," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[54] D. Yang *et al.*, "Automatic liver segmentation using an adversarial image-to-image network," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017.

[55] P. F. Felzenszwalb and D. P. Huttenlocher, “Efficient graph-based image segmentation,” *Int. J. Comput. Vis.*, 2004.

[56] A. Neubeck and L. Van Gool, “Efficient non-maximum suppression,” in *Proceedings - International Conference on Pattern Recognition*, 2006.

[57] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 2015.

[58] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[59] S. V. Stehman, “Selecting and interpreting measures of thematic classification accuracy,” *Remote Sens. Environ.*, 1997.

[60] D. M. W. POWERS, “Evaluation: From Precision, Recall and F-Measure To Roc, Informedness, Markedness & Correlation,” *J. Mach. Learn. Technol.*, 2011.

[61] C. Buckley and E. M. Voorhees, “Evaluating evaluation measure stability,” in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’00*, 2000.