VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

TRAN TUAN CANH - 517H0099

AUTOMATED MEASUREMENT OF FETAL HEAD CIRCUMFERENCE USING 2D ULTRASOUND IMAGES

UNDERGRADUATE THESIS OF COMPUTER SCIENCE

Advised by

PhD. LE MINH HUNG

HO CHI MINH CITY, YEAR 2021

VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

TRAN TUAN CANH - 517H0099

AUTOMATED MEASUREMENT OF FETAL HEAD CIRCUMFERENCE USING 2D ULTRASOUND IMAGES

UNDERGRADUATE THESIS OF COMPUTER SCIENCE

Advised by

PhD. LE MINH HUNG

HO CHI MINH CITY, YEAR 2021

ACKNOWLEDGEMENT

During the past time, I could not have been done without the kind support from many individuals and organizations – which is why I would like to send my most profound gratitude to all of them.

First of all, I would like to express a special appreciation to Ton Duc Thang University's teachers for their conscientious guidance, we has learned a lot of useful things and accumulated some knowledge to complete into this report.

I am highly indebted to Mr. Le Minh Hung for his guidance and constant supervision as well as for providing necessary information regarding the project and also for their support in completing the project. Again, I sincerely thank you.

I would like to express my special gratitude and thanks to industry persons for giving me such attention and time.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

Ho Chi Minh City, May 10, 2021

Author

(Signature and full name)

Tran Tuan Canh

DECLARATION OF AUTHORSHIP

I hereby by declare that this thesis was carried out by myself under the guidance and super vision of PhD Le Minh Hung; and that the work and the result contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures in this thesis are for analysis, comments, and evaluations from various resources by my own work and have been duly acknownledged in the reference part.

In addition, other comments, reviews and data used by other authors, and organizations have been acknownledged, and explicitly cited.

I will take full responsibility for any fraud detected in my thesis. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

Ho Chi Minh city, May 10, 2021

Author

(Signature and full name)

Tran Tuan Canh

EVALUATION OF INSTRUCTING LECTURER

Confirmation of the instructor

....
....
....
....
....

Ho Chi Minh city, day month year

(Full name, Signed and Sealed)

The assessment of the teacher marked

....
....
....
....
....

Ho Chi Minh city, day month year

(Full name, Signed and Sealed)

Abstract

In this paper, we introduce a segmentation system that can automatically measure the fetal head circumference (HC) in real-time by using ultrasound images. With the optimized results, the HC can be applied to estimate the gestational age and keep track on the health of the infants. Therefore, it can help solving the shortage of well-trained specialists, doctors, and sonographers.

The system includes three core stages that are responsible for specific task. The first stage is a detection and segmentation stage which adopts convolutional neural network to generate a mask for the fetal head. After that, we optimize the result of the mask by using an ellipse fitting algorithm that smoothing the edge of the mask. Finally, in the third stage, a simple ellipse perimeter calculation is applied to estimate the HC. The system was trained, and tested on 999 ultrasound images and 335 ultrasound images respectively.

The model effectively shows a robust performance with the detection loss and segmentation loss are 0.0129 and 0.0656 respectively in validation stage. Thus, it outperforms many traditional systems that employ handcrafted methods to estimate the HC.

Contents

List Of Abbreviations

HC	Head Circumference
CRL	Crow-Rump Length
GA	Gestational Age
MLP	Multi Layer Perceptron
SVM	Support Vector Machine
CNN	Convolution Neural Network
GD	Gradient Descent
SGD	Stochastic Gradient Descent
R-CNN	Region Based Convolutional Neural Network
RPN	Region Proposal Network
FPN	Feature Pyramid Network
IoU	Intersection over Union
FC layer	Fully Connected layer
FCN	Fully Convolutional Network
acronym	full name

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Reason for choosing topic

In the modern world of medical field, ultrasound images are the most broadly used to monitor and diagnosis fetus during pregnancy. Many women prefer to use it to follow the development of fetus because it is economical, real-time monitoring, and extremely safe compared to X-rays or others types of imaging systems that use radiation based-method. Ultrasound examinations provide parents with a valuable opportunity to understand the health status of their unborn child. While ultrasound is considered to efficient and safe, it still has several disadvantages.

The process of ultrasound imaging depends mostly on the one who operates the system for its signature noises on image such as shadows and reverberations which makes the image extremely hard to observe and diagnose the fetus' status. That is the reason why ultrasound system can only be used by well-trained, professional specialists, doctors, and sonographers which leads to the shortage human resources in poor and developing countries.

1.2 Target implementation

In the fetal ultrasound image, ultrasound allow the visualization of some body features, possibly other parts such as fingers and toes of the fetus. Based on these important features, many biometric measurements are applied by doctors or ultrasound imaging operators. For example, the crow-rump length (CRL) and head circumference (HC) are commonly calculated to estimate the gestational age (GA) and given diagnosis about growth of the fetus. The CRL widely uses for its accuracy to estimate the GA of the fetus in the age between 8 weeks 4 days and 12 weeks. After 13 weeks, specialists usually use HC for its accuracy instead of CRL. The instruction of HC measurement state that HC should be measured in a transverse section of the head with a central midline echo, interrupted in the anterior third by the cavity of the septum pellucidum with the anterior and posterior horns of the lateral ventricles in view [7].

The HC measurement steps are proceeded manually which make the result unstable when being conducted by different doctors. The idea of creating an automated system is born base on the above obstacle. With the support from computer, the HC measurement result will be no longer affected by observer variability along with the shortage of sonographers.

Chapter 2

Related Works

2.1 Introduction to traditional approaches

In the past decades, many systems for automatic HC measurement have been introduced with various traditional approaches. In 2005, Wei Lu et al presented a system that used a low-pass filter to reduce noise in ultrasound images and a transformation filter to increase contrast between skull and background (this process method is based on the signature of bones on ultrasound image that usually brighter than the background). And they optimized K-mean algorithm and morphologic binary area opening operation to achieved skull segment. After that, Wei Lu et al assumed the skull was elliptical shape and then applied an iterative randomized Hough transform to predict the offset value of the ellipse. Their result was quite remarkable with the differences between sonographers and the system only 0.52% while predicting HC [14].

In 2017, Jing li et al was inspired by ellipse fitting methods, and proposed a system that employed a random forest algorithm to locate the fetal head. Then, they used phrase symmetry algorithm to detect the ellipse center and applied a non-iterative ellipse fitting method to efficiently fit the ellipse on the fetal head. As a result, their method archived an average measurement error of 1.7 mm and outperformed traditional methods [12].

In the next year, Thomas L. A. van den Heuvel et al introduced a method included three systems that measure the HC in all trimester of pregnancy. Each system architecture has different number of pipelines that employs a random forest algorithm to extract Haar-like features and a set of Hough transform, dynamic program, ellipse fitting algorithm to measure the HC. Therefore, by focusing on the nature of each trimester, this system showed not only the feasibility but also the robustness on fetal heads of each trimester [7].

2.2 Conclusion and determine the problem

Many systems for automatic HC measurement have been introduced with various traditional approaches using Hough transform, Haar-like features, ellipse fitting, etc. giving promising results. However, variations on the dataset are high due to noise, format, screening parameters configurations, etc. Therefore, traditional methods which based on hand-crafted features are not robust to all the variations of the images.

Base on others research, we consider this problem is not only the detection task but it is, specifically, the segmentation task that separates the fetal head from the noisy background. In this research, we focus on a more novel method for fetal head detection and segmentation using convolutional neural networks (CNNs) that show the robustness on a lot of computer vision tasks [23], [8], [22].

Many models in this research have been utilized to evaluate the performance of different CNNs configuration to analyze the insight of the dataset. The CNN model we used was fine-tuned and trained on 999 ultrasound images, and then it was tested on 335 ultrasound images of the HC-18 dataset which is public on *grand-challenge.org*. The model significantly shows robustness on the dataset with the detection loss and segmentation loss are 0.0129 and 0.0656 respectively in validation stage.

Chapter 3

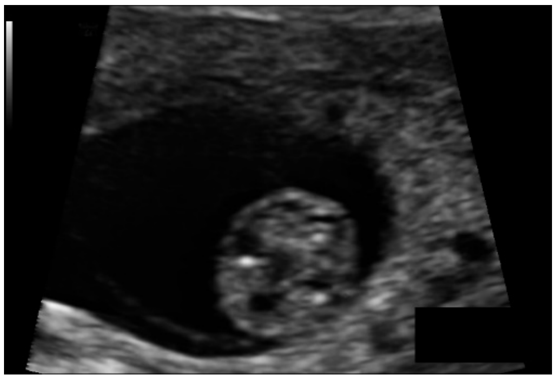
Materials And Methods

3.1 HC18 Dataset

3.1.1 Description

The collection of 1334 2D ultrasound images of fetal head were collected from the database of the Department of Obstetrics of the Radboud University Medical Center, Nijmegen, the Netherlands. All of the fetal head ultrasound images for this challenge were captured in the standard plane which is the specifically used to measure the HC [7].

The data is divided into a training set of 999 images and a test set of 335 images. The size of each 2D ultrasound image is 800x540 pixel with a pixel size ranging from 0.052 to 0.326 mm. The information of pixel of each image can be found in the CSV files: 'training-set-pixel-size-and-HC.csv' and 'test-set-pixel-size.csv'. The variability of pixel size was resulted from the adjustment of sonographers during examinations which led to a different shape and size of the fetal heads. In addition, in the training set, come along with each ultrasound image is a manual annotation in millimeters. All of the image file-names start with a number. However, the file-names only set to 805 because some images were come from the same examination (different frame during monitoring), therefore they have a similar appearance [7].


(a) A fetal head ultrasound image.

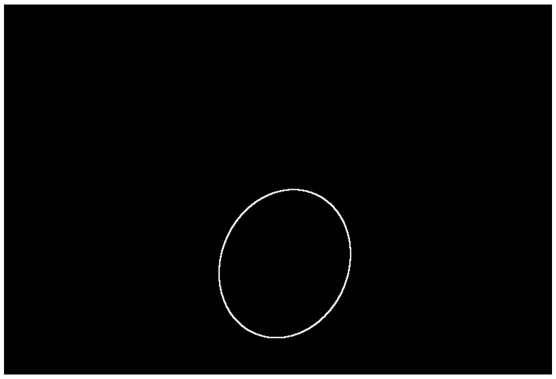

(b) An annotation created by sonographers.

Figure 3.1: HC-18 dataset sample.

3.1.2 Pre-processing

During each examination, the sonographer manually draws an ellipse which best fit the fetal head and save it as annotation of the ultrasound image. And based on these annotations, we apply a few image processing to extract the ellipse coordinates and created a dataset in COCO format for further training methods.

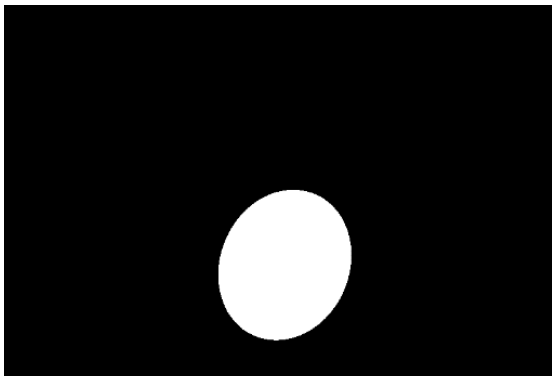

(a) a processed annotation

Figure 3.2: This annotation is made for Mask R-CNN model.

And then we save it in json file - ”annotations.json” with the information structure including images, categories, and annotations.

Images:

- File name and ID.
- Height and width: size of each image.

Categories:

- Super-category: name for group of objects (for example: fish – which include lots of fish species).
- ID and name

Annotations:

- Segmentation: coordinates of annotated pixels (contour).
- Area: ground true binary mask.
- Is crowd: one or multiple objects in the image.
- Image ID, ID, category ID.
- Bounding box: offset values.

3.2 Convolutional neural network

3.2.1 A brief history of CNN

In the age of information and technology, the hardware resources have made some extraordinary breakthrough which enhances the parallel calculation power of both CPU and GPU. Based on these success, artificial intelligence field is back to the race after decades being hold back because of the limitation of hardware.

In the recent year, “Deep Learning” has become the most popular keywords in artificial intelligence field. However, its history was date back to the late 1950s with the invention of perceptron architecture. Being influenced by the capability of biological system on perceptual recognition, generalization, recall, and thinking, F. Rosenblatt et al tried to mimic a brain functions by focusing his research on the smallest element of the brain - neurons.

The author claimed that perceptron's design could simulate several signature features of the intelligence systems at an acceptable standard - a better-than-chance probability without applying multiple techniques to create a special environment of a specific biological creature. As a result, Rosen blatt believed that the fundamental laws of all physical system and humans could eventually be understood [16].

Even though perceptron showed a promising future for the intelligence system, the problem perceptron could not encounter was XOR. To be more specific, XOR operation is a linear un-division operation which cannot be represented by a single-layer perceptron. Hence, multi-layer perceptron (MLP) was adopted to solve the issue [24]. However, it was extremely hard and ineffective to train the MLP back then.

Therefore, not until the 1980s, MLP with backpropagation algorithm was finally introduced as an upgrade of the original perceptron that solve the image classification problems [1]. These algorithms achieved a few successes but then being constrained because computer at that time was not strong enough to run such heavy model (not mention the quality and quantity of the data).

After the birth of Support Vector Machine (SVM) algorithm which solved lots of disadvantages of perceptron models, deep learning once again being forgotten [cite SVM]. Not until 1998, Yann LeCun developed the model called LeNet, one of the first convolutional neural networks which has 2 layers (Convolution + Max pooling) and 2 fully connected layers and softmax layer as the output layer. Yann LeCun's model marked the comeback of deep learning after achieved significant accuracy (up to 99%) on MNIST dataset [11]. After the success of LeNet, lots of models had been developed and achieved promising results on image classification problem.

In 2012, another milestone was the winner of ImageNet LSVRC-2012 - AlexNet of

Geoffey Hinton et al. This model was the breakthrough in deep learning field which opened a new era for the revolution of neural network and contributed directly to a numerous artificial intelligence application recently. AlexNet is a huge model with 5 convolutional layer and 3 fully-connected layer (around 60 million parameters) was trained on ImageNet dataset which has around 1.2 million images of 1000 labels. The network achieved a top-5 error of 15.3, more than 10.8% points lower than its rivals. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of graphics processing units (GPUs) during training [10].

Since 2012, researchers around the world have started to create a numerous model from wider ones to deeper ones and achieved countless breakthrough and knowledge in deep learning field. Some outstanding models such as VGGNet (Karen Simonyan, Andrew Zisserman et al 2014), GoogleNet (Szegedy et al 2014), ResNet (Kaiming He et al 2015), etc., have significant performance on computer vision tasks that allows computers work at human level.

VGG16 model was first introduced by a team of researchers from Oxford University in 2015. It was developed based on the goal of increasing the depth of the model. Therefore, compared with LeNet, instead of using the larger kernel size (5x5 or 11x11), VGG model take the advantage of 3x3 kernel to enable the ability to learn a more complex patterns in the dataset and also decrease the computational cost [18].

In the same year of 2015, another model call GoogLeNet was proposed and also archived result as good as VGG16. In this paper, the author introduced a brand new concept - inception block. It is a dense structure module that includes a few convolution kernel of size 1x1, 3x3, and 5x5 which helps reduce the computational power. Therefore, it enables the model to be deeper than any counterparts before.

Base on these state of the art models, plenty more architectures have been proposed to solve problems from many different fields. They are not only getting more and more accurate and lighter, but they are also optimized to execute multi-tasks on real-time processing (which is hardly to achieve in the past) and solve a vast majority works of human.

3.2.2 A CNN architecture

In the previous section, we introduce briefly about CNN history, so what exactly is it? A typical CNN architecture consists of 3 types layers which are the input layer, the hidden layers, and the output layer. Unlike the traditional neural networks which only have layers of fully-connected neurons following by activation functions, CNNs also have layers that perform convolution, pooling layer to down size the feature map, and then the fully-connected layers.

3.2.2.1 Convolution layer

In the context of CNN, a convolution is a linear operation that involves the multiplication a set of weights with the input. The multiplication is performed between an input array and a weights array called filter or kernel. Let's say every image's pixel is a value in a 2D matrix. Convolution layer are the major building blocks used in neural network. Each of convolution layer has a specific number of filters which slide over the image to extract its features by execute the dot product. A dot product is the element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the "scalar product" [25].

For example:

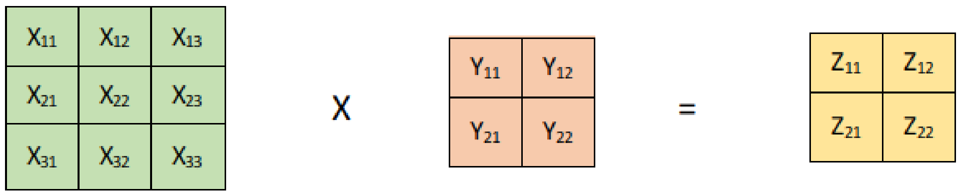

X_{11} X_{12} X_{13}
X_{21} X_{22} X_{23}
X_{31} X_{32} X_{33}
X
Y_{11} Y_{12}
Y_{21} Y_{22}
=
Z_{11} Z_{12}
Z_{21} Z_{22}

Figure 3.3: Example of convolution operation.

Calculation step as below:

$$Z_{11} = X_{11}Y_{11} + X_{12}Y_{12} + X_{21}Y_{21} + X_{22}Y_{22}$$

$$Z_{12} = X_{11}Y_{12} + X_{12}Y_{13} + X_{21}Y_{22} + X_{22}Y_{23}$$

$$Z_{21} = X_{11}Y_{21} + X_{12}Y_{22} + X_{21}Y_{31} + X_{22}Y_{32}$$

$$Z_{22} = X_{11}Y_{22} + X_{12}Y_{23} + X_{21}Y_{32} + X_{22}Y_{33}$$

This process seems simple, but it is very effective in image processing. In a CNN, the input is a tensor with shape (number of images) x (image height) x (image width) x (input channels). After passing through a convolution layer, the image becomes abstracted to a feature map (the result after applying convolution operation), with shape (number of images) x (feature map height) x (feature map width) x (feature map channels).

To train or modify a CNN, there are some attributes we should keep track on:

- The size and the number of filters/kernels (hyper-parameters).
- The number of input and output channels (hyper-parameters).
- The quantity of filters is equal to the quantity of feature maps.
- Padding and Stride attributes in convolution operations.

Padding and Stride are the crucial technique in convolution. Let's take a look at the previous example in 3.3. We calculated the feature map by sliding the filter on the input data by one pixel for each convolution operation, we refer to the number of rows and columns traversed per slide as the stride. The smaller step we define in filters, the more information we can extract from it, but it is a trade-off which affect the training time. One tricky issue when applying convolution layers is that we tend to lose pixel information on the perimeter of input data.

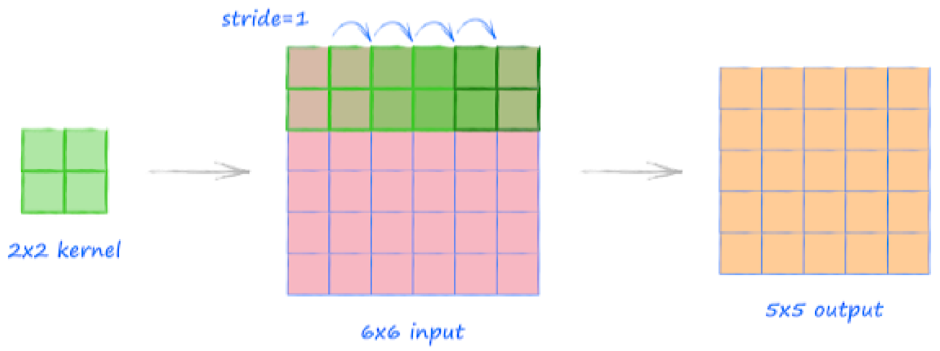


Figure 3.4: Missing pixels information when doing convolution.

This problem happens when we slide the filter on input data, our filters only do convolution operation on pixels at the edge of matrix fewer than pixels which are close to the center of the matrix. Even though, it is just a few pixels, we use lots of convolution layers, it will add up and cause missing information. That is why the term “Padding” comes up, a direct solution for this is to add a “barrier” of zero pixels surrounding the input matrix which increases the effective size of the input matrix.

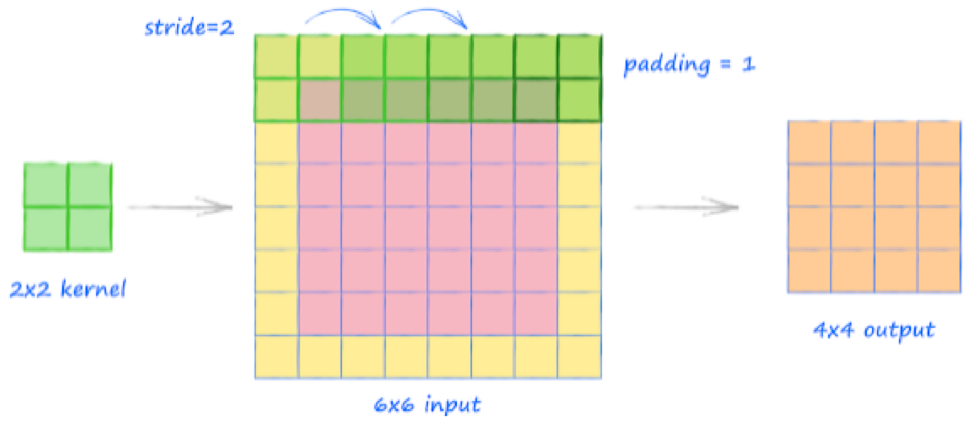


Figure 3.5: Applying zero padding = 1 to the input matrix.

3.2.2.2 Pooling layer

As described in previous section, the convolutional layer output is a feature maps, as it can be regarded as the learned representations (features) in the spatial dimensions (e.g., width and height) to the subsequent layer. Usually, as we process an image, we want to reduce the spatial resolution of our feature maps, select information so that it can be more robust or sensitive when we go deeper in the network. By gradually aggregating information, yielding coarser and coarser maps, our model can accomplish a global representation for the feeding dataset.

Like convolution, pooling operator consists of a fixed-shape window that slide all over on the output feature map according to its stride setting. However, unlike sliding window of convolutional layer, sliding window of pooling layer does not have any parameters (no kernel). Instead of calculating the feature map, pooling layer simply calculate either the maximum or average value of the pixels inside its sliding window. These operations are called maximum pooling and average pooling, respectively. Another characteristic of pooling layer is that it also can alter padding and stride settings to prevent model from missing information.

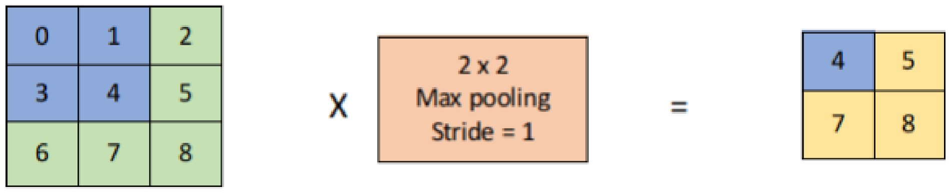


Figure 3.6: Max pooling with window shape 2x2 and stride = 1.

3.2.2.3 Fully connected layer

Fully-connected (FC) layers connect every neuron in one layer to every neuron in another layer. It is the similar to a traditional multi-layer perceptron neural network (MLP). After we extract feature maps from convolutional layers, we flatten these maps to one dimension vector and then feed it to the FC layer for the model to learn how to classify the data. However, FC layers can be seen as a brute force approach whereas there are approaches like the convolutional layer which reduces the input to concerned features only. That is the reason why CNN models are often built with few FC layers (one or two layers).

3.2.2.4 Loss function

In the training process, a CNN model learns to map a set of inputs to a set of outputs. By using gradient-based training method, the mapping process cannot be calculated perfectly in one step due to convergence process. Therefore, the model has to feed the inputs from the dataset over and over again to improve the mapping result. As a result, the problem of learning is explore as optimization problem.

Base on this idea, loss function (or cost function) is defined as the difference of the dataset output and the predicted output of the model. It is like a form of getting the model to pay a penalty after a wrong prediction, and loss function output is proportional to the severity of the error. In supervised learning problem, the goal is to minimize this loss as low as possible (In the ideal condition, loss function returns 0).

A general loss function is expressed as below:

$$L_\varepsilon\Big(y, f(x,w)\Big) = \begin{cases} 0 & \Big|y - f(x,w)\Big| \leq \varepsilon, \\ \Big|y - f(x,w)\Big| - \varepsilon & \text{otherwise,} \end{cases}$$

As loss function calculate the difference between the right and the wrong mapping value, naturally, it can be defined as a subtraction of the two values. However, subtracting 2 values may result in negative number which cannot be considered, therefore, an absolute expression is added to constrained its output.

$$L(\hat{y}\, y) = |\hat{y} \ - y|$$

Because the model is trained based on gradient based method, the minimization process of above function is hardly to achieve due to the un-continuous derivative (for example, derivative of $f(x) = |x|$ is interrupted at 0). To address this issue, the whole expression is squared and then divide by two.

A simple loss function:

$$L(\hat{y}\,y) = \frac{1}{2}\,(\hat{y}\ - y)^2$$

Let's take a binary classification problem to be an simple example which $\hat{y}\ < 0$ means the model prefers -1 prediction and $\hat{y}\ > 0$ means the model prefers 1 prediction. Hence, an appropriate loss function that satisfies the following criteria:

1. Whenever the model make a wrong predict, it must be punished badly. Therefore, the first criteria is that loss function has to return a bigger value if $\hat{y}$ is opposite to y than the contrast.

2. If there are two answers $\hat{y}$ and y both have the same sign (or different sign), which one should be punished more? As mentioned before, the absolute value $|\hat{y}\ |$ represents the "prefer-ness" of the model for an option. The larger this value, the more the model prefers an answer. In the case of the same sign, the preferred option is the correct one, so the more the model likes its prediction, the less penalties must be paid. With the same argument, if the sign $\hat{y}$ is different from y, because the preferred option is the wrong one, the more the model likes, the more severe penalties will be imposed on the model so that the model will not repeat it.

Yet, with different problems, there are different loss functions with multiple criteria. Therefore, before applying any models, algorithms, researchers need to identify the problem so that they can choose a suitable loss function to train the model.

3.2.2.5 Optimization function

In the previous section, we discussed about the loss function and its criteria. Once we define a loss function to make penalty on the wrong prediction of the model, we need to minimize the loss value in attempt to make the model prediction better after each training step. Therefore, an algorithm called optimization algorithm was brought into the training process in order to serve the optimization problem in the training phrase.

Although optimization provides a way to minimize the loss function for deep learning, in essence, the goals of optimization and deep learning are fundamentally different. The former is primarily concerned with minimizing an objective whereas the latter is concerned with finding a suitable model, given a finite amount of data.

Despite the fact that optimization algorithm provides a method for deep learning model to minimize the loss function value, in general, the purpose of them is essentially different from one another. In deep learning field, we mostly concern find a model that can generalize a finite amount of given data. However, the goal of optimization is to minimize the objective function.

To be more specific, training error and generalization error are ordinarily different:

1. As the objective function of optimization algorithm is often a loss function, so its goal is to decrease the error in the training process.

2. In deep learning or statistical field, the goal is to reduce generalization error.

To accomplish the latter we need to pay attention to over-fitting in addition to using the optimization algorithm to reduce the training error [25].

There lots of problems when we try to optimize loss function of a deep learning model. Therefore, in this section, we will focus on several controversial issues such as local minima, saddle point, gradient exploding and gradient vanishing.

As mentioned in loss function section, gradient-base method is mostly apply to train a deep learning model using backpropagation algorithm. Hence, in the process of training a model, the derivative calculation is done continuously and throughout to find the global minimum point of the function representing the data set - $f(x)$. For ease of visualization, we will look at 3.7.

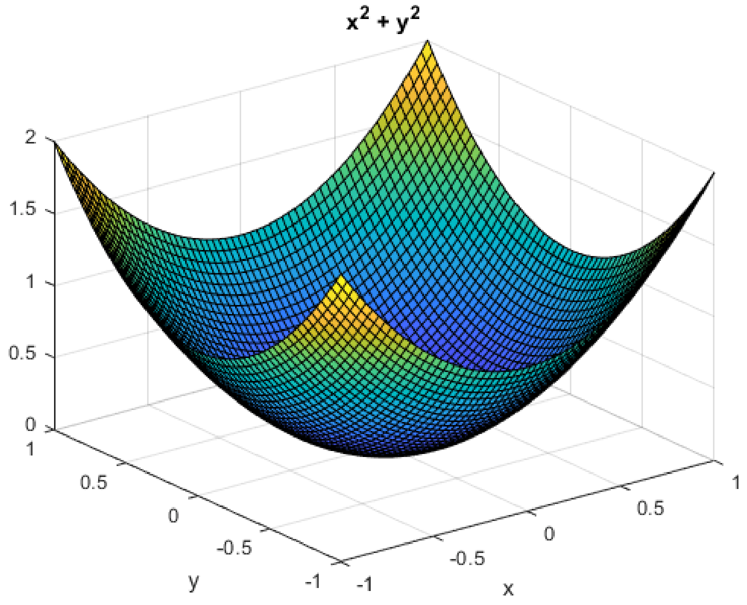


Figure 3.7: Bowl shape function - $x^2 + y^2$.

Assuming this cup-shaped plane is the equation f (x), its minimum point will be in the center of the cup. If we jump into this plane, of course we will slip straight to its minimum point. So when a function converges at its minimum, it optimizes as much as possible in the ability to classify the data. This is the fundamental goal of training: to continue modifying weights until the global minimum has been reached.

However, not all functions are cup-shaped. Let's take a look at 3.8 below for easier visualization.

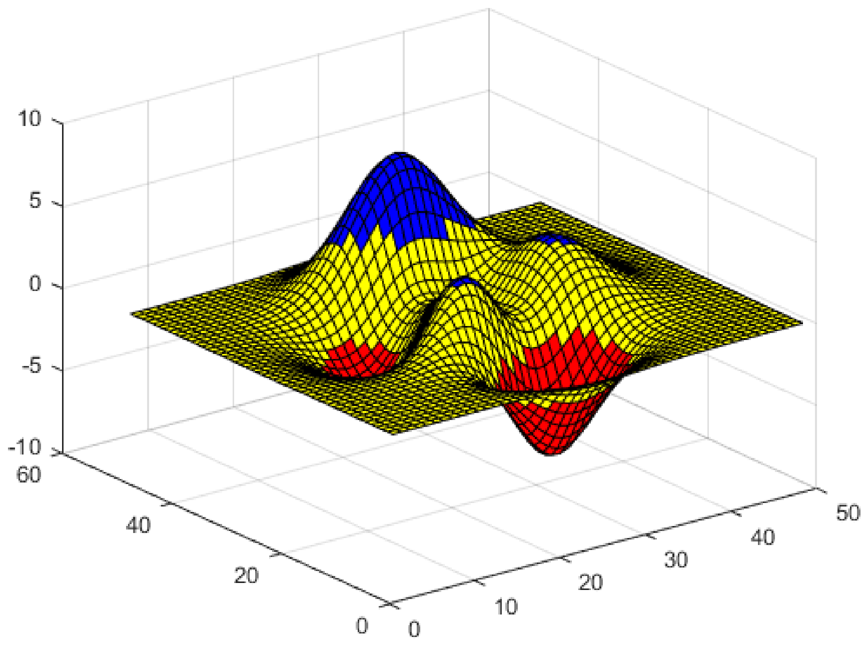


Figure 3.8: Local minima and global minima.

A function of the deep learning model usually takes the same form for many local minima points. Then, the function is susceptible to convergence at local minima points because it can only optimize the function of the model locally. This causes the gradients to be zero early and adversely affects the model's training and its results.

Besides local minima, saddle points are also a cause which make gradient vanish early. Although saddle point can cause gradient vanishing, it is not neither local minima nor global minima.

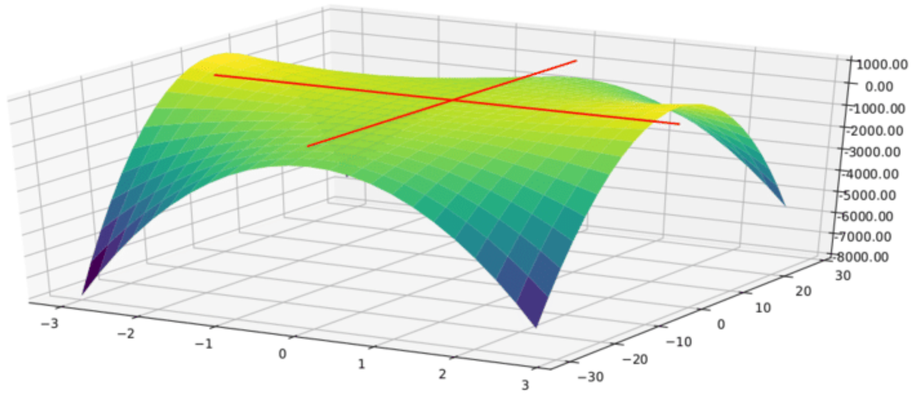


Figure 3.9: Saddle point.

3.3 Gradient descent

In deep learning in particular and optimization math in general, we often have to find the smallest value of a function. However, finding the global minima of loss functions in deep learning is very complicated or even impossible. Instead, people often try to find the local minima points, and to a certain extent, consider it the solution to be found in the problem.

The local minima points are the solutions of the derivative equation equal to 0. If it is possible to find the whole (finite) minimum points, we can replace each of those local minimum points into the objective function and then find the point making it the smallest value.

Yet, in most cases it is not possible to solve zero derivative equations. The cause can come from the complexity of the derivative form, from the fact that the data points have a large number of dimensions (Look at 3.10 for more detail).

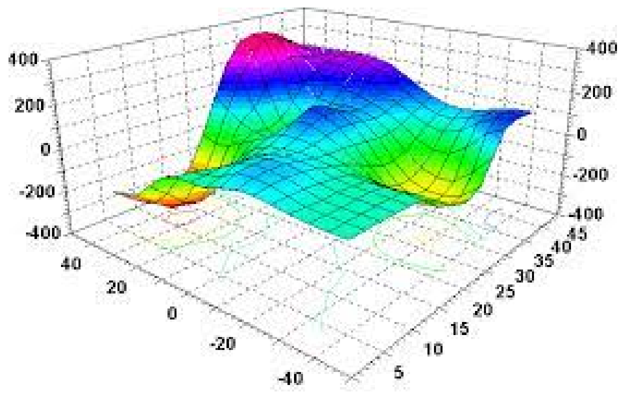


Figure 3.10: It is hard for GD to be converging at global minima.

The most typical approach is to start from a point that we consider to be close to the global minima, then use an iteration to move closer to the desired point. Gradient Descent [17] and its variations are among the most popular methods.

In this part, we will take gradient descent in one dimension as a simple example to explain it functional mechanism. Let assume we have a function as follow.

$$f(x) = \frac{1}{2} \cdot (x-1)^2 - 2$$

Supposed x_t is the point that we find after the t-th iteration. The goal is to define a function that can bring x_t as close as possible to the global minima after a specific number of iteration.

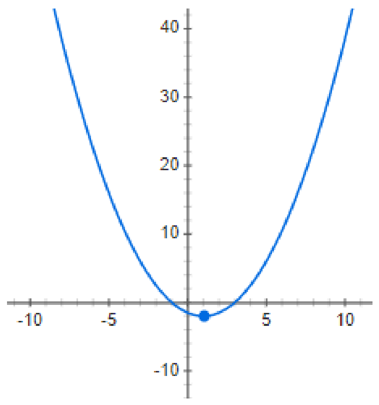


Figure 3.11: $f(x) = \frac{1}{2} \cdot (x-1)^2 - 2$.

Look at the plot of the objective function. It is noticeable that if the derivative of the objective function at x_t is greater than zero, x_t will stay on the right compared to the global minima and contrast. Therefore, to bring x closer to the goal, it is needed to be moved toward the left side of the minima which means the negative side. In other word, x_t has to move oppositely to the derivative.

$$x_{t+1} = x_t + \Delta$$

Δ is a quantity opposite to the derivative $f'(x)$
The farther away x_t is from x to the right, the larger $f(x_t)$ and vice versa. Therefore, delta is proportional to $-f'(x_t)$. As a result, the equation is represented as following:

$$x_{t+1} = x_t - \eta \cdot f'(x_t)$$

Where η is a positive number called learning rate. The minus shows that it has to move oppositely to the derivative. It is the reason why this method is called Gradient Descent (means moving backward). In conclusion, the constructing of the function above, even though it is not always reasonable for all problems, it is a foundation for lots of optimization approach in general and deep learning in particular.

3.4 Multivariate gradient descent

Let say we need to find global minima for the function $f(\theta)$ that θ is a vector usually used to denote the set of parameters (or weights) of the objective function. The derivative of the function at any θ is denoted as $\nabla_\theta f(\theta)$. It is similar to one variable function, GD for multivariate is also started at a predicted point θ_0. After that, it is updated as following:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t)$$

Or it is simply represented as:

$$\theta = \theta - \eta \nabla_\theta f(\theta)$$

Note: Always move oppositely to the derivative.

3.5 Gradient descent variants

In general, there are in total three variants of GD algorithm. Each of them has has a unique approach on using the amount of data to calculate the derivative of the objective function. Therefore, the amount of data used will affect directly on the convergence speed of the derivative and may cause a trade-off between the accuracy and the time on each update of the weights.

3.5.1 Batch gradient descent

The GD algorithm is mentioned from the beginning of GD section is the vanilla one - batch gradient descent [17]. In this circumstance, the word ”batch” means ”all” which means all of the data x_i are used whenever an update is perform on the weights $\theta = w$. Therefore, this technique shows that computation in training process is effectively due to no update required after each training sample.

This vanilla approach works ineffectively if there is a huge dataset. Because it has to calculate gradient and update weights for all data in the dataset after each epoch which slowdown the training process.

3.5.2 Stochastic gradient descent

In deep learning, the objective function is usually the average of the loss functions for each example in the training dataset. By using SGD [17], at a time, only one data point x_i is used to calculate gradient and then it updates the weights - θ. This work is executed with each data point in the whole dataset and repeated after each epoch.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x_i; y_i)$$

$J(\theta; x_i; y_i)$ is a loss function with the input is a pair data (input, label).

With normal GD, each epoch corresponds to 1 update θ, but with SGD, each epoch corresponds to N update θ (N is the number of data points in the dataset). Look at one hand, updating is executed on each data point may lead to a slow training epoch. Look at the other hand, SDG only requires a few epoch to complete its training process. Therefore, SGD is suitable for a huge dataset rather than the vanilla one.

Yet, due to the frequent updates, the steps taken towards the minima are very noisy. This can often lean the gradient descent into other directions or may not settle on global minima. Moreover, SGD is not only expensive in computation because of using all resources to train, but it also loses the advantage of vectorized operations as it deals with only one data sample at a time

3.5.3 Mini-batch gradient descent

Similar to SGD, mini-batch GD [17] uses n data points (greater than 1 and smaller than the whole dataset). Mini-batch GD starts each training epoch by shuffle all data points and divides them into small batches. Each of them has n data points (except for the last batch may have fewer if N is not divisible to n) At each update, mini-batch GD takes one batch to calculate gradient and updates it as following.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x_i; y_i)$$

By taking the advantages of both original GD and SDG, mini-batch GD shows its computational efficiency (compared to the vanilla GD), stable convergence towards global minima, and faster learning as it calculate gradient on multiple data samples and update the weights more regularly (compared to SGD).

As a result, mini-batch GD has been applied widely in machine learning field, especially in deep learning.

3.6 Momentum

In section 3.5.2, SGD illustrates that in the optimization process, noisy gradients may lead to convergence suspend or even fail to converge at a good minima. In this section, we will introduce a effective technique that can help us overcome the problem in practice - momentum [17].

The gradient descent algorithm sometimes is compared to the effect of gravity on a ball on a surface as figure..., so eventually the ball will settle at the global minima. However, with the surface as figure..., the question is how the ball can overcome the local minima to settle at the global minima. The answer is quite simple if we solve this problem more physics-ly.

Physically, if we consider the changed-quantity at the t time to update a new position for the ball as a velocity quantity so the formula for updating is $\theta_{t+1} = \theta_t - v_t$. At the moment, our problem is to calculate v_t so that it not only has the gradient information but also has the momentum (which is the previous velocity - v_{t-1}). Naturally, we calculate the summary of both quantity.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta f(\theta)$$

Therefore, the ball has a chance to overcome the local minima and move forward by utilizing the momentum quantity.

3.7 RMSprop

Unlike previous discussed optimization algorithms that their learning rate α stay unchanged in the training process (learning rate is a constant). RMSprop [17] is an unpublished approach proposed by Geoffrey Hinton et al in a course on Coursera. The author considers learning rate as a parameter that can change throughout the training process after each time t. It is demonstrated as following formula:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

RMSprop optimization divides the learning rate by the exponentially decreasing mean of the squared gradients. Hinton suggests γ is set to 0.9, while a good default for learning rate η is 0.001.

3.8 Adam

Adaptive moment estimation called Adam [17] [9] is an adaptive learning rate optimization algorithm that has been developed specifically for training a deep learning neural network. It was designed in 2014 and introduced at ICLR 2015. Adam is a combination between RMSprop (Geoffrey Hinton at el) and SGD with momentum. Specifically, Adam calculates adaptive learning rate for each parameters individually and store the exponentially decay average of past squared gradients v_t and m_t like RMSprop and momentum, respectively.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

As illustrated, Adam uses the squared gradients to scale down the learning rate like RMSprop and takes the advantage of momentum by using the average of gradient instead of gradient itself like SGD. m_t and v_t are estimated the value of the mean and the uncentered variance (meaning it does not subtract the mean during variance calculation).

To execute the estimation, Adam utilizes the exponentially moving averages using the gradient that evaluated on the current mini-batch where m and v are the moving averages, and g is the gradient on present mini-batch.

Adam update rule is shown as following.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t$$

The good default value for β_1 is 0.9 and β_2 is 0.999, and 10^{-8} for ε proposed by the author himself.

Chapter 4

Region Based Convolutional Neural Network

4.1 The problem of region of interest

Computer vision is a large field that has been attracted lots of researchers, scientists in the recent years (after AlexNet's milestone). Beside image classification problem, another research trend of computer vision is object detection. The difference between two types of problem is that classification problem wants to know the label or labels of the image, while detection problem try to locate a bounding box surrounding the object inside the image. The challenge is there could be many bounding boxes representing different objects and we do not how many beforehand. The major reason why typical CNN models as we describe in section 2 cannot solve this problem is that the length of the output layer is not constant (unknown number of objects). A naive approach for this is that we can take different region of interest (RoI) and apply CNN model to check whether the RoI has object. Yet, this approach also has a huge problem is that the RoIs may have different spatial locations in the image, therefore, it will cost an expensive computational resource because of a huge number of RoIs.

4.2 The original R-CNN

In 2014, a new model had been proposed named region based convolutional neural network. The model has a remarkable record on PASCAL VOC 2010 with the mean average precision (mAP) of 53.7% and mAP of 31.4% on ILSVR2013 detection dataset [4].

To solve the issue of the numerous Rois, Ross Girshick et al. proposed a method where we use selective search to extract just 2000 regions from the image and he called them region proposals [4]. As a result, instead of blowing up the computational resource to classify RoIs like the naive approach, now we can just work with 2000 regions.

According to the author, their system consists of three modules. The first module is the one that solve the expensive computation issue by using selective search to generate category-independent region proposals. The second module is a CNN that extract a 4096-dimensions feature vector from RoIs as the output. The last one is a set of SVMs and a linear regression models to classify the object and predict its bounding box, respectively.

Problems with R-CNN:

- Training is a multi-stage pipeline and expensive in space and time.
- It cannot be implemented as real time system because it takes 47s/image to extract features from RoIs when using VGG16 as backbone (on a GPU).
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

4.3 The improvement in Fast R-CNN and Faster R-CNN

Next year, in 2015, Ross Girshick et al again proposed a new model called Fast R-CNN which solved the drawbacks of their previous model. This model architecture looks similar to the original R-CNN but instead of feeding the RoIs to CNN, an input image and multiple regions of interest (RoIs) are input into a CNN. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers

(FCs). The network has two output vectors per RoI: softmax probabilitiesand per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss [3].

Fast R-CNN has several advantages according to the paper:

- Higher detection quality (mAP) than R-CNN.
- Training is single-stage, using a multi-task loss.
- Training can update all network layers.
- No disk storage is required for feature caching.

With Fast R-CNN, we do not have to feed 2000 RoIs to CNN which enhance the training and testing speed by execute the convolution once per image and feature map will be generated from it. The network is illustrated in figure 6 (below).

Both of the above models (R-CNN & Fast R-CNN) use selective search to generate the RoIs. However, selective search is a slow and time-consuming process affecting the performance of the network [15]. Therefore, Shaoqing Ren et al came up with an object detection algorithm that alternates the selective search algorithm and allows the network completely learn to propose the RoIs and they called it Region Proposal Networks (RPNs). By sharing convolutional layers, the marginal cost for computing proposals is significantly decrease (e.g., 10s/image) [15].

As an upgrade from Fast R-CNN, the new model was introduced in 2016 by Shaoqing Ren et al called Faster R-CNN that improves the speed of the model towards real-time object detection. This new architecture is similar to its predecessor that image is provided as an input to a convolutional network to generate feature maps. Yet, instead of applying selective search, a separated network called RPNs (which is completely trained) is used to predict RoIs. In the next stage, the predicted results are reshaped using a RoIs pooling layer. And then, the rectangular RoIs are used for classifying the objects inside the image and predicting the offset value of bounding boxes.

Model	Describe	Testing speed	Limitations
CNN	Divides the image into multiple regions and then classify each region into various classes.	unknown	Naive approach take a huge number of regions to predict, therefore, expensive computation cost.
R-CNN	Apply selective search, choose 2000 regions to process separately.	47s on VOC07	Expensive computation cost from difference CNN models for each region
Fast R-CNN	Extract feature maps first then apply selective search to generate regions.	0.32s on VOC07	Selective search is slow and un-trainable.
Faster R-CNN	Alternate selective search by RPN which can be trainable and much faster.	0.2s on both COCO and VOC07	Object proposal takes time and as there are different systems working one after the other, the performance of systems depends on how the previous system has performed.

Table 4.1: Comparison between CNN, R-CNN, Fast R-CNN, and Faster R-CNN

4.4 Region proposal network (RPNs)

In object detection using region based technique, RPN is a one true backbone which takes an image as the input and outs a set of rectangular RoIs. Although the RPN can process a whole image, the ultimate goal of the author is to share computation with the Fast R-CNN. As the result, the RPN is modified to be a small network that works as the second stage in the model architecture and it take the convolutional feature map output by the last shared convolutional layer of Faster R-CNN object detection network as the input. And then, this small network slide a n x n spatial window over the feature map to map it from high dimensional feature to lower one. This feature then is fed to two siblings fully-connected layers - a box-regression layer and a box-classification layer [15].

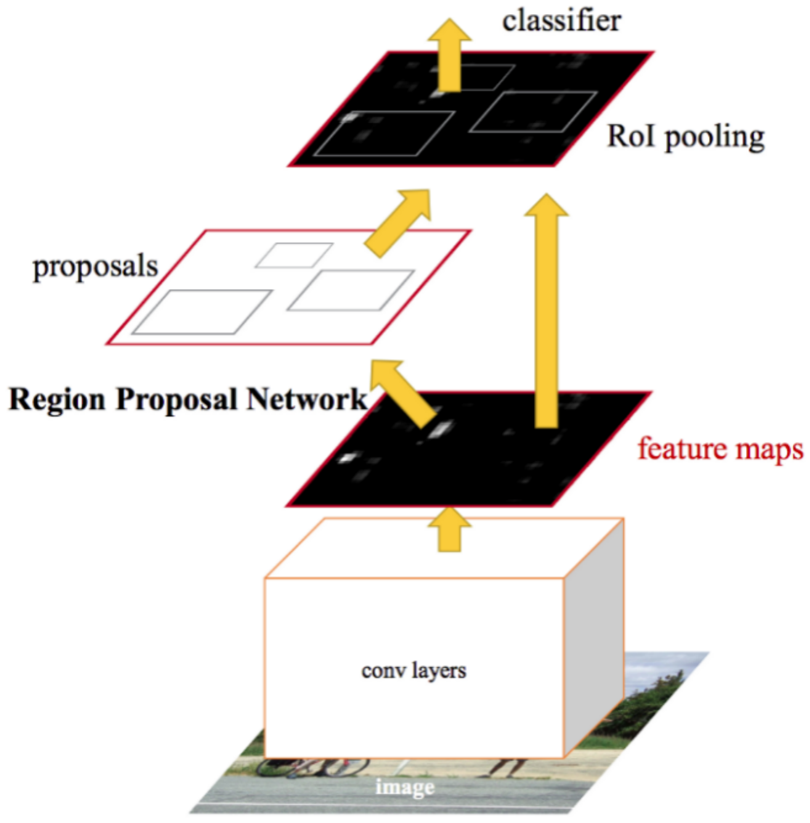


Figure 4.1: Faster R-CNN architect including RPN.

4.4.1 Anchors

At each sliding window position, we predict simultaneously lots of proposals with k is denoted as the maximum number of proposals at each location. According to the paper, the regression layer has 4k predicted coordinates of k boxes and classification layer has 2k predicted probability outputs of object or not object for each proposals. Therefore, the k value helps the authors parameterize the unknown number of boxes to k boxes and they called them anchors. An anchor is defined as the centered boxes of each sliding window. Base on the paper, the authors used 1 square, 2 rectangles with 3 scales and 3 aspect ratio to create 9 anchors (so k = 3 x 3 = 9). Therefore, with a feature map of the size W x H there are WHk anchors in total. These anchors are labeled with positive or negative base on the area that overlap with the ground truth box as the following rule:

- The anchor is positive when it is the maximum value of intersection over union (IoU) and it is bigger than 0.7.
- The anchor is negative when it is smaller than 0.3.
- The anchor that neither positive nor negative does not consider to be the training objective.

4.4.2 ROI pool layer

For each object proposal, RoI pooling layer extracts a fixed length of feature vector from the feature map so that it can be fed into the classifier and regressor in the final FC layer. More explicitly, the RoI pooling can be described in three steps:

- Dividing the region proposal into equal-sized sections (the number of which is the same as the dimension of the output).
- Finding the largest value in each section.
- Mapping these max values to the output buffer.

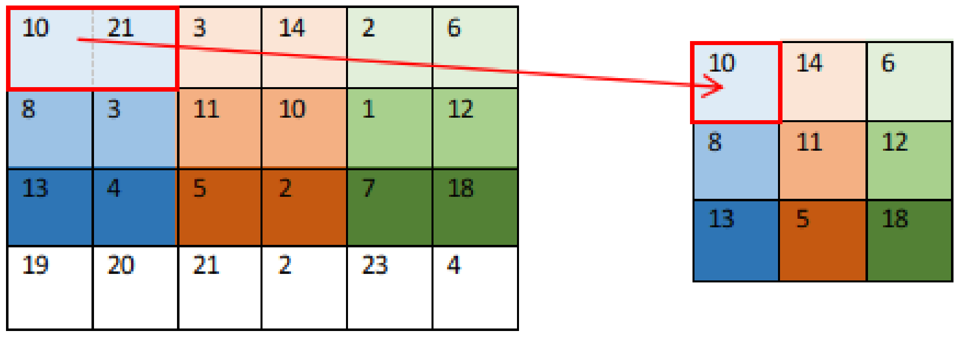


Figure 4.2: An example of RoI pooling window size 3x3 on vector 4x6.

As we can see on 4.2, RoI pooling only take 1x2 vector to map its maximum value to a fixed length vector because quantization process ($4/3 \sim 1.3333 = 1$, $6/3 = 2$). Therefore, the entire bottom row does not take in to account causing missing information of the feature map. This process will directly affect the result accuracy.

Chapter 5

Mask R-CNN

5.1 The problem with Faster R-CNN

Until now, we know how Faster R-CNN extract feature maps by passing the image through lots of convolutional layer. However, while down sampling the image, we also scaled down the RoIs inside the image with a specific factor and round the offset of their bounding boxes. As a result, we create a new bounding box for the object inside which cause missing information and reduce performance of our system.

An example 5.1, it illustrates when we feed 512x512 image to VGG16 and cause quantization on bounding box offset. The blue part is the missing piece data and the one green is a new data created by quantization.

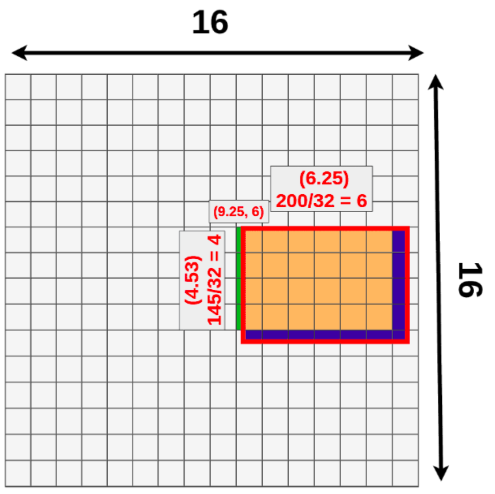


Figure 5.1: Example of quantization problem causing missing information.

As mention before in 4.4.2, RoI pooling operation will quantize floating number of RoI offset to a discrete offset. Then this quantized RoI is subdivided into spatial bins which are then aggregated (usually by max pooling) [15]. And once again, we lose vector information. We can look at 5.1 for more details.

5.2 An extended solving the problem - Mask R-CNN

In 2017, a group of Facebook AI researchers - Kaiming He at el presented a new system which is influenced by Faster R-CNN called Mask R-CNN. This system, gennerally, is an extend of Faster R-CNN with a new branch for segmentation generating an object mask. Unlike others system which are complex multiple-stage cascade that predicts segment proposals from bounding-box proposals, followed by classification. Mask R-CNN allows three branches (including the existing branch for classification and bounding box regression, and a branch for predicting segmentation masks on RoI) to run in parallel which enhance the processing speed.

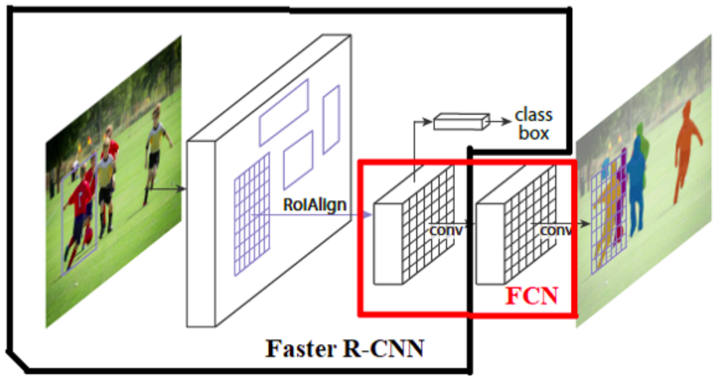


Figure 5.2: Mask R-CNN architecture including Faster R-CNN and FCN

As 5.2 shows that Mask R-CNN is constructed by Faster R-CNN and a small fully convolution neural network. Because it is a stack of convolutional layer, the mask branch only consumes a small computational resource enabling the system to run extremely fast in testing phrase just like Faster R-CNN speed (0.2s/image).

It is noticeable that Mask R-CNN's authors solved the problem of RoI Pooling quantization by proposing a brand-new alternative technique called RoI Align which completely secure spatial locations of bounding box and information inside it. According to the paper, this minor change can affect the result accuracy up to 50%.

5.2.1 Multi-task loss

Inherit the spirit of Faster R-CNN, Mask R-CNN also has two stage procedure, with RPN is the first stage that generates proposals. In the next stage, in parallel with the original branch of Fast R-CNN, there is a mask generation branch that outputs a binary mask for each RoI. Therefore, a new loss function was proposed and defined as Ł = $L_{cls} + L_{box} + L_{mask}$. The classification and regression loss (L_{cls}, L_{box}) are the same as those defined in [3].

The mask loss, according to the authors, was defined as the average binary cross entropy loss (per-pixel sigmoid and binary loss) enabling the system to generate masks for each class without causing competition among them. As a result, lots of masks will be generated but only the ground truth masks of class k in the corresponding RoIs are considered while calculating the mask loss (masks on another classes are not contribute to mask loss of class k). And by taking advantage of the classifier branch, the predicted class will be used to choose the output mask. This strategy decouples the mask and class prediction and outputs good instance segmentation results. Therefore, Mask R-CNN's strategy is different from most techniques at that time (in semantic segmentation problems) when adopting a per-pixel multinomial logistic loss and validate with the standard metric of mean pixel intersection over union (IoU), with the mean taken over all classes - softmax (competition between classes), including background [5].

$$L_{mask} = -\frac{1}{m^2} \sum_{1 \leqslant i,j \leqslant m} [y_{ij} log \hat{y}^k_{ij} + (1 - y_{ij}) log(1 - \hat{y}^k_{ij})]$$

5.2.2 Mask representation

Different from label and offset branch that collapse the feature map into a fixed-vector by using FC layer. Mask branch generates a mash that represents the object area in the input image. To be more specific, it encodes object's spatial information to a binary map that separates the object from the background. To achieve this goal, naturally, Mask R-CNN adopts a FCN to extract pixel-to-pixel information by taking advantage of convolution operations.

So instead of adopting a FC layer to classify each pixel and generate mask from it, the FCN uses fewer parameters, and is more efficient than the FC layer. However, to preserve mask generating's efficiency, RoI must be extracted without losing minimum information so that the spacial structure correspondence to original input image will be maintain accurately. Therefore, RoI align layer plays a crucial role in Mask RCNN [5].

5.2.3 RoI align

As mentioned before in 4.4.2 and 5.1, RoI pooling's quantization causes missing information two times in it process. Firstly, a quantization operations is executed to map the generated RoI's coordinates (x, y indexes) from floating values to integer ones. Secondly, this quantized RoI is then sub-divided into small boxes whose size are rounded up. As a result, these quantization operations lead us to the problem of misalignment between the RoI and the extracted feature map. Even though, it does not affect the classification or detection task due to the robustness on small adjustment, it is extremely sensitive on the pixel predicting of the segmentation task and impact directly to the model performance.

To address the problem, Kaiming He and his partners proposed a method called RoI align that erase the harassment of quantization from RoI pool technique. The change is minor but it works significantly effective on segmentation task. By taking advantage of bi-linear interpolation, the whole quantization issue is solved. Similar to RoI pool method, the RoI is divided into pre-fixed number of smaller regions. Within each smaller regions, 4 points are sampled. To be more specific, the calculation does not fall on a specific pixel, and the nearest pixel is used to perform bilinear interpolation on this *virtual pixel* to obtain the pixel value and then max or average operation is executed to get the final result [5].

5.2.4 Mask R-CNN architecture

According to the original paper, the architecture of Mask R-CNN is divided into 2 parts which are the backbone and the head.

- The backbone is a convolutional network used for extracting feature maps from the input image.
- The head is the part that includes three parallel branches for object detection, bounding box regression, and mask generation.

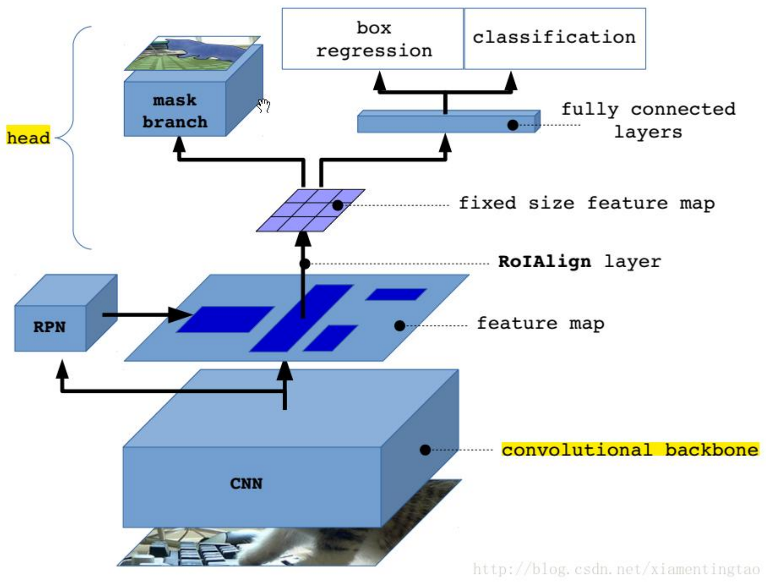


Figure 5.3: The head and the backbone of Mask R-CNN architecture

In our work, we train and evaluate ResNet network of depth 50 and 101 layers [6] combine with Feature Pyramid Network proposed by the authors of Mask R-CNN. By taking advantage of two state of the art networks, we not only can extract the feature that is more robust on the dataset, but we also can extract them in different levels according to the FPS scale [5].

For the network head, a fully convolutional mask prediction branch is added in parallel with two branches of the Faster R-CNN's head. Moreover, the authors also optimize Mask R-CNN with fewer filters when using FPN which already includes the the res5 backbone. Look at Figure 5.4 for more details.

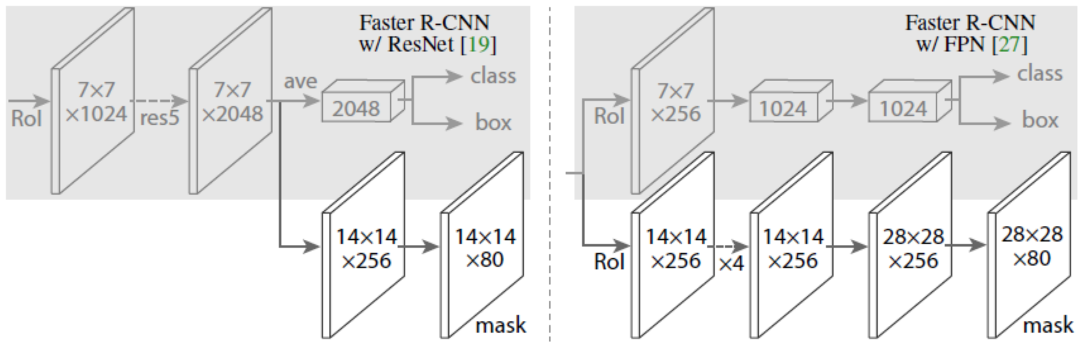


Figure 5.4: Head architecture of ResNet (left) compares to FPN (right)

For further information of Mask R-CNN's architecture, we recommend readers to [5].

Chapter 6

Implementation and Evaluation

6.1 Training Mask R-CNN

In this research, we take advantage of a pre-trained model of Mask R-CNN and fine-tune it to for our problem instead of training the whole architecture of Mask R-CNN from scratch due to the lack of time and resources. Although Facebook's researchers published the official source code in 2018, we prefer using the unofficial one from Matterport Inc which was published in 2017 due to the majority of deep learning practitioner community (5k3 fork times compared to 9k6 ones, respectively). Further information, you can access the source code via this link `https://github.com/matterport/Mask_RCNN`.

In addition, the whole training process was executed on Google's CoLab to take advantages of GPU power (which we did not have). Besides, the training loss and the validation loss, we also calculated mAP and mAR to evaluate the models more precisely.

6.1.1 Configuration setting

In the training phrase, focused on fine-tune several hyper-parameters that affected strongly for transfer learning technique. According to [5], the default configuration was set based on Faster R-CNN due to its robustness on not only on detection task but also segmentation one. Look at Table 6.1 for more details about hyper-parameters.

Model	Describe
IMAGE RESIZE MODE	Input image resizing.
STEPS PER EPOCH	Number of training steps per epoch. Don't set this too small to avoid spending a lot of time on validation stats.
VALIDATION STEPS	Number of validation steps to run at the end of every training epoch. A bigger number improves accuracy of validation stats, but slows down the training.
BACKBONE	Backbone network architecture. ResNet50 and ResNet101 are supported.
LEARNING RATE	Learning rate.
LEARNING MOMENTUM	Momentum
OPTIM	Optimization algorithm.
WEIGHT DECAY	Weight decay regularization.
LOSS WEIGHTS	Loss weights for more precise optimization. (Focus on desired loss)

Table 6.1: Hyper-parameter description

In Table 6.1, we only list several hyper-parameters that are important in the fine-tune process. In addition, for more information of hyper-parameters, we recommend reader to read at `https://github.com/matterport/Mask_RCNN/blob/master/mrcnn/config.py`.

6.1.2 Training

As mentioned before, we used transfer learning technique to train Mask R-CNN. The dataset we used was described in 3.1.1. With a small pre-processing, we changed the annotation images and exported annotation's information (COCO's format) from it to fit Mask R-CNN's input requirement. In addition, we also applied data augmentation to increase the dataset size in order to avoid over-fitting problem. To be more precise, we created more images by:

- Flipping 50% images vertically and horizontally, respectively.
- Rotating all images by 90, 180, and 270 degree.
- Multiplying images by a random value sampled uniformly from the interval [0.8, 1.5], making some images darker and others brighter.
- Blurring images using a gaussian kernel with a random standard deviation sampled uniformly (per image) from the interval [0.0, 5.0].

In the training process, we took advantages by using a pre-trained of Mask R-CNN trained on COCO dataset.... INTRODUCE the SCORE. Because we only trained the head of the network, we decided to use 20-30 epochs.

The result is illustrated in Table 6.2, 6.3. All of these configurations were set and fin-tuned based on [5] and `https://github.com/matterport/Mask_RCNN`.

	Test 1	
Backbone	Resnet50	Resnet50
Optimizer	SGD	Adam
Learning rate	0.001	0.001
Weight decay	0.0005	0.0005
Momentum	0.9	0.9
Epoch	20	20
Train - Val step	150 - 50	150 - 50
Detection min confidence	0	0
Detection NMS threshold	0.9	0.9

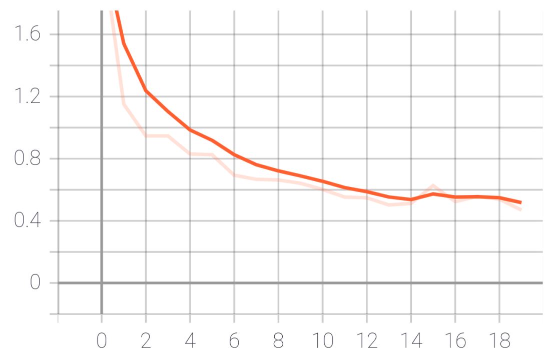
RPN anchor scale	16, 32, 64, 128, 256	16, 32, 64, 128, 256
Resize mode	square	square
Loss weight	rpn class loss: 1.0,	rpn class loss: 1.0,
	rpn bbox loss: 1.0,	rpn bbox loss: 1.0,
	mrcnn class loss: 1.0,	mrcnn class loss: 1.0,
	mrcnn bbox loss: 1.0,	mrcnn bbox loss: 1.0,
	mrcnn mask loss: 3.0	mrcnn mask loss: 3.0
mAP	0.511111	0.822222222
mAR	0.533333	0.837037037
NOTE	SGD train faster but trade off accuracy	Adam shows better much

	Test 6	**Test 7**	**Test**
Backbone	Resnet50	Resnet50	Resnet50
Optimizer	Adam	Adam	SGD
Learning rate	0.0001	0.0001	0.001
Weight decay	5,00E-05	5,00E-05	0.0005
Momentum	0.9	0.9	0.9
Epoch	30	30	30
Train - Val step	150 - 50	150-50	150 - 50
Detection min confidence	0.7	0.7	0.7
Detection NMS threshold	0.7	0.7	0.7
RPN anchor scale	16, 32, 64, 128, 256	16, 32, 64, 128, 256	16, 32, 64, 128, 25

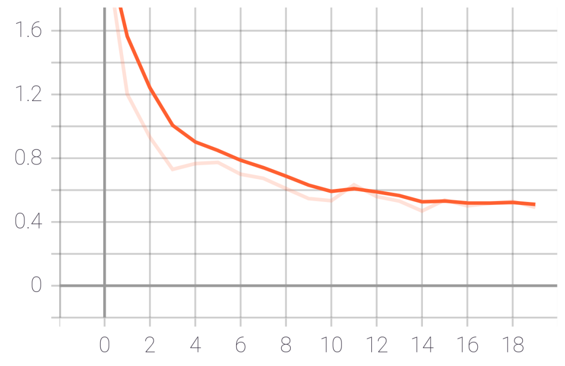
	Test 6	**Test 7**	**Test
Resize mode	square	square	square
Loss weight	rpn class loss: 1.0,	rpn class loss: 1.0,	rpn class loss: 1.0,
	rpn bbox loss: 1.0,	rpn bbox loss: 1.0,	rpn bbox loss: 1.0,
	mrcnn class loss: 1.0,	mrcnn class loss: 1.0,	mrcnn class loss: 1
	mrcnn bbox loss: 1.0,	mrcnn bbox loss: 1.0,	mrcnn bbox loss: 1
	mrcnn mask loss: 1.0	mrcnn mask loss: 1.0	mrcnn mask loss: 1
mAP	0.43703703703	0.955555555556	0.16296296298
mAR	0.62222222222	0.970370370370	0.16296296298
NOTE	???	???	SGD not works we

Table 6.3: Experiment results

Firstly, we started our training process with ”Test 1” and ”Test 2” in order to find out which optimizer was better than the other. In general, it is noticeable that Adam optimizer converged slightly faster than SGD one. Moreover, model with Adam optimizer showed a much better mAP and mAR compared to model with SGD. Yet, the total loss, especially the mask loss was still high (more than 0.4 and 0.2, respectively). Let’s look at Figure 6.1 and 6.2 for more details.

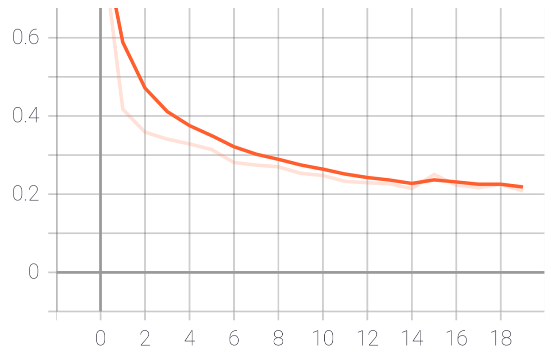


(a) Total loss in Test 1.

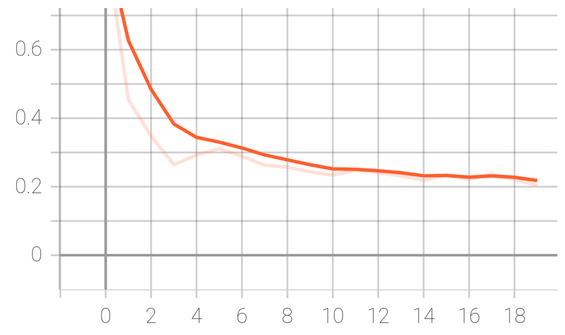


(b) Total loss in Test 2.

Figure 6.1: Comparison between Test 1 and Test 2 loss.



(a) Mask loss in Test 1.



(b) Mask loss in Test 2.

Figure 6.2: Comparison between Test 1 and Test 2 mask loss.

Noted: In training phrase, we mostly paid attention for the mask loss due to the characteristic of the problem which was instance segmentation. Therefore, we altered the weight balance with the ratio was 1.0 - 1.0 - 3.0 (for classification loss, detection loss, and segmentation loss, respectively) in order to calculate the perimeter of HC with the minimum error using the generated mask.

In next the experiments, we change the backbone from ResNet50 to ResNet101 to see could we actually improve the model by going *deeper*? And the results were quite significant. In ”Test 3”, with ResNet101, we improved the model mAP and mAR from 0.5 to 0.97777 which was the highest score in lots of experiments we had proceeded. However, the total loss of ”Test 4” went to NaN due to the high learning rate which showed Adam optimizer was quite sensitive to the high learning rate (overshooting problem). Therefore, in ”Test 5” we decreased the learning rate to 0.0001, consequently, we could boost mAP and mAR score to 0.9419753 and 0.9703703, respectively.

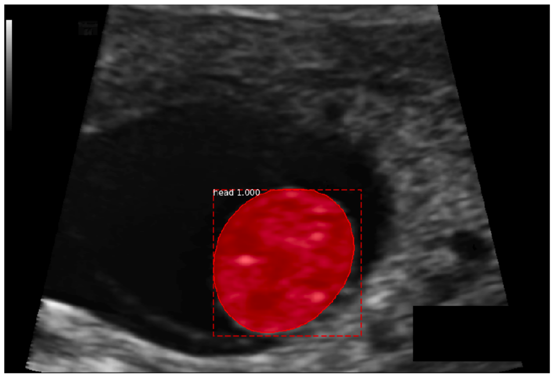
Although, Adam optimizer with ResNet101 backbone also showed a remarkable result (Test7, 10), SGD optimizer with ResNet101 backbone illustrated the improvement in results of ”Test 3” and ”Test 9” with mAP and mAR are (0.977777, 0.977777) and (0.9407408, 0.9851852), respectively.

6.1.3 Inference

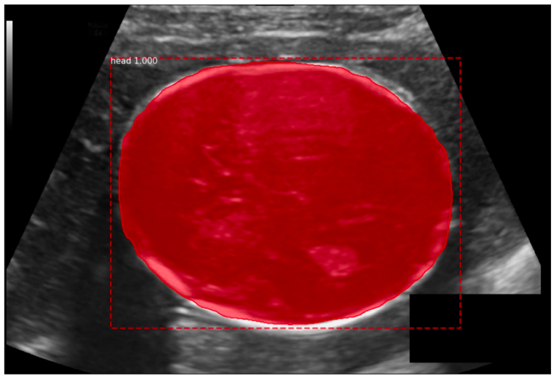
The testing phrase was executed on the PC with hardwares as following:

- CPU Intel Core i5 8th.
- RAM 12GB.

Even with this minimum system, Mask R-CNN could generate mask with around 1 2 second for each testing image. The result was shown in Figure 6.3.



(a) Generated mask on training sample.



(b) Generated mask on testing sample.

Figure 6.3: We can see that segmentation part on testing sample were not perfect as training one. Therefore, the perimeter was estimated from this mask would be affected.

As we can see from Figure 6.3, the generated mask was fit perfectly on the fetus' head. Yet, the model was not robust enough on the testing image (we can still see the white part of the skull that had not covered yet by the mask).

6.2 Building the Restful API

In this section, we will introduce our proposed system that includes five components includes 2 parts to transfer and manage the input, and 3 parts to process HC images and estimate the perimeter of the fetus' head. Each of the components is an restful API which works on specific task (inspired from micro service design). Therefore, in the future, we can easily scale up the number of workers (API) to speed up the process and manage them more conveniently in the deployment phrase. For more details, the system's design is shown in 6.4.

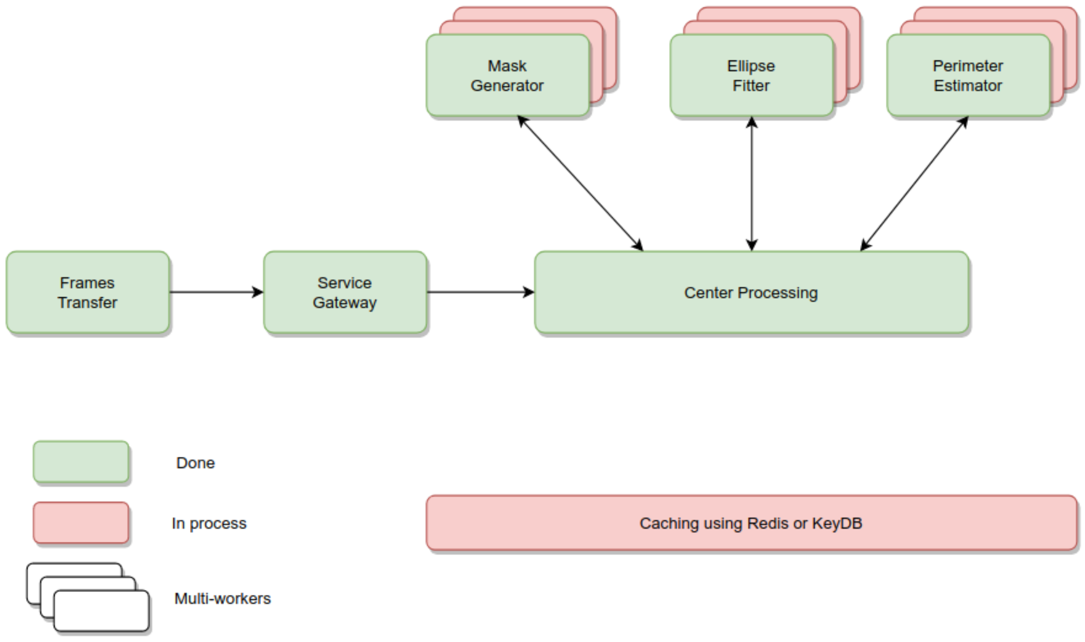


Figure 6.4: Proposed demo HC estimator system.

6.2.1 Service Gateway

Briefly, service gateway is an API that receives fetus' head from multiple sources such as images, video, or even streaming form. We decided to separate it to be standalone in order to process multiple transfer types more conveniently. However, in this thesis, this system is simply a demo so it only supports image transfer which is sent through a HTTP request.

More precisely, the input data for Service Gateway are pixel size (to calculate the perimeter) and image of fetus' head in RGB color channel 3.1.1, .

6.2.2 Center Processing

This API, in short, it is like a CPU that sends data and receives result to/from others APIs. To be more specific:

- Firstly, it sends a image to Mask Generator, and then receive a binary mask.
- Secondly, it sends the binary mask and the image to Ellipse Fitter, and then receive an ellipse that fits on the mask.
- Finally, the ellipse's information is processed by Perimeter estimator to calculate the circumference of the ellipse (which is the fetus' head).

Besides, it works as a manager to monitor, report, etc.. the process of others APIs. Moreover, because we plan to increase the number of workers for specific task as in Figure 6.4, it is necessary to have a center processing to control the work flow of the system.

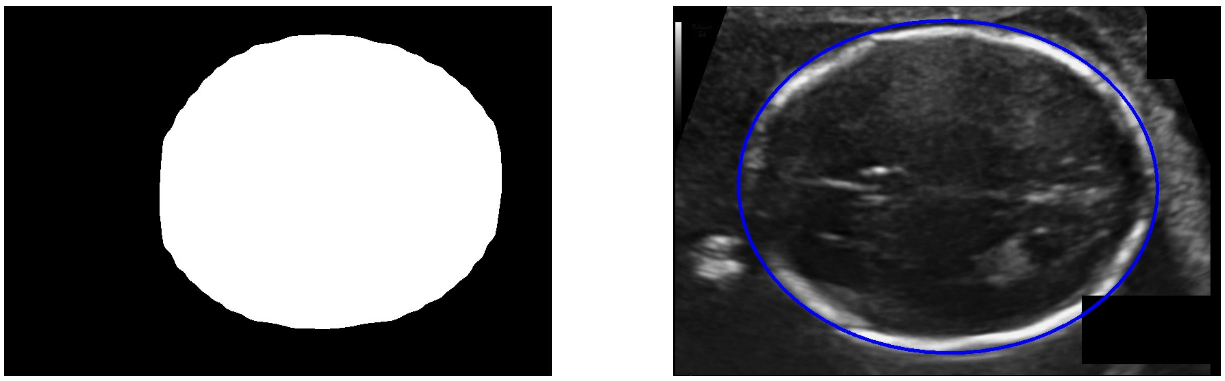
6.2.3 Mask Generator

As the first processing stage, Mask Generator receives a request from Center Processing to segment the fetus's head in an RGB image and returns a binary mask for it. Base on previous experiments, we decide to use the weights from ”Test 3” (which is our best model) to build Mask Generator API despite its imperfection in inference stage (but the result is still acceptable) as Figure 6.3 shows.

Even though the whole mask generating process is worked on a non-GPU computer, it still maintains the processing speed at 1 2s per image. Therefore, with a better hardware, we believe that the processing speed will definitely decrease and may achieve real-time processing speed to a certain extent.

6.2.4 Ellipse Fitter

After generating mask for the fetus' head, the Contour-finding and the Ellipse-specific algorithms are applied to adjust the polygon shape of the mask to the ellipse shape as in Figure 6.5.



(a) The binary mask. (b) The ellipse boundary on the fetus' head.

Figure 6.5: Combine finding contour algorithm with ellipse fitting one to it an ellipse on the image.

Precisely, a curve joining of all continuous points (along the boundary of the binary mask) is found by using algorithm from [19]. Despite this method is simple to implement through OpenCV, it works quite efficiently on our binary mask.

For an ellipse, its function requires a set $\chi = (x^2, xy, y^2, x, y, 1)$. However, we only need five parameters to define an ellipse on 2D space.

- (x, y): center x, center y.
- (MA, ma): semi axes a, semi axes b.
- Angle radian

To achieve this goal, we fit the ellipse around this set of 2D points by employing the Direct Least Square approach from [2].

6.2.5 Perimeter Estimator

After receiving the ellipse's coordinates from Ellipse Fitter API, we estimate the ellipse's perimeter by employing Ramanujan's formula [21]. The approximation of Ramanujan is demonstrated as in Equation (6.2.5):

$$h = \frac{(a-b)^2}{(a+b)^2}$$

$$p \approx \pi(a+b)(1 + \frac{3h}{(10+\sqrt{4-3h})})$$

Then we multiply the p with the pixel size to scale the result to the exact measure.

6.2.6 Result

To evaluation the whole system, we tested it with 335 ultrasound images and then we submitted the result to Grand Challenge. As the result, we were ranked at 1077 position with several metrics as follow:

DICE:

- "max": 27.589732124216823,
- "min": 76.75946495249799,
- "std": 3.7815463871178943,

- ”25pc”: 83.50825381011883,
- ”50pc”: 85.81469956490609,
- ”75pc”: 88.01129094018415,
- ”mean”: 86.09170119968213,
- ”count”: 335.0

Difference:

- ”max”: 27.589732124216823,
- ”min”: -0.6480596018764686,
- ”std”: 4.33538798698353,
- ”25pc”: 4.394918205609386,
- ”50pc”: 6.114941307429092,
- ”75pc”: 8.74297314561646,
- ”mean”: 7.147501055171521,
- ”count”: 335.0

DICE trimester 1:

- ”max”: 97.13935203256567,
- ”min”: 76.75946495249799,
- ”std”: 4.762801395468964,
- ”25pc”: 85.62838488544139,
- ”50pc”: 88.8016218891177,

- ”75pc”: 92.7723271301243,
- ”mean”: 88.74547608698568,
- ”count”: 55.0

DICE trimester 2:

- ”max”: 97.0691841623221,
- ”min”: 77.47039916485811,
- ”std”: 3.3353566574286297,
- ”25pc”: 83.26670659930204,
- ”50pc”: 85.55823051459525,
- ”75pc”: 87.29931334288395,
- ”mean”: 85.63891977747174,
- ”count”: 233.0

DICE trimester 3:

- ”max”: 90.40598274885838,
- ”min”: 78.04261888610793,
- ”std”: 3.298341158642274,
- ”25pc”: 83.15553019467893,
- ”50pc”: 85.53754670022215,
- ”75pc”: 87.85791957902292,
- ”mean”: 85.23085976507201,

- ”count”: 47.0

Hausdorff distance:

- ”max”: 20.589144777463485,
- ”min”: 0.7724761587361798,
- ”std”: 3.583340499351782,
- ”25pc”: 5.516364786956328,
- ”50pc”: 7.548516888575833,
- ”75pc”: 9.419397962500073,
- ”mean”: 7.581017919690942,
- ”count”: 335.0

Absolute difference:

- ”max”: 27.589732124216823,
- ”min”: 0.6480596018764686,
- ”std”: 4.3289838125839815,
- ”25pc”: 4.394918205609386,
- ”50pc”: 6.114941307429092,
- ”75pc”: 8.74297314561646,
- ”mean”: 7.151370067720037,
- ”count”: 335.0

Difference trimester 1:

- ”max”: 15.111908868158608,
- ”min”: 1.2812882510039572,
- ”std”: 2.9241650358535636,
- ”25pc”: 3.134473332958102,
- ”50pc”: 4.602075402152096,
- ”75pc”: 6.087198198104765,
- ”mean”: 5.135097830934518,
- ”count”: 55.0

Difference trimester 2:

- ”max”: 24.749099690650667,
- ”min”: -0.6480596018764686,
- ”std”: 3.579054028927821,
- ”25pc”: 4.464146812138978,
- ”50pc”: 6.026485291046242,
- ”75pc”: 7.730095196420621,
- ”mean”: 6.635340857744364,
- ”count”: 233.0

Difference trimester 3:

- ”max”: 27.589732124216823,
- ”min”: 0.6867050791128122,

- ”std”: 5.584556464260941,
- ”25pc”: 9.190157286174411,
- ”50pc”: 11.662282500140009,
- ”75pc”: 14.501014051515938,
- ”mean”: 12.041447934609026,
- ”count”: 47.0

Hausdorff distance trimester 1:

- ”max”: 6.097752780066,
- ”min”: 0.7724761587361798,
- ”std”: 1.1635430749332072,
- ”25pc”: 1.98026176953945,
- ”50pc”: 2.7580496387675,
- ”75pc”: 3.6799445304227216,
- ”mean”: 2.8235145949238367,
- ”count”: 55.0

Hausdorff distance trimester 2:

- ”max”: 14.658116046154605,
- ”min”: 1.0214043471768,
- ”std”: 2.045828977818262,
- ”25pc”: 6.316217417726442,

- ”50pc”: 7.616355581727,
- ”75pc”: 8.87503939518,
- ”mean”: 7.535175687838671,
- ”count”: 233.0

Hausdorff distance trimester 3:

- ”max”: 20.589144777463485,
- ”min”: 8.011363070117723,
- ”std”: 3.0255606034609572,
- ”25pc”: 11.575047545355172,
- ”50pc”: 12.37692592766892,
- ”75pc”: 15.573101160804221,
- ”mean”: 13.37556947040946,
- ”count”: 47.0

Absolute difference trimester 1:

- ”max”: 15.111908868158608,
- ”min”: 1.2812882510039572,
- ”std”: 2.9241650358535636,
- ”25pc”: 3.134473332958102,
- ”50pc”: 4.602075402152096,
- ”75pc”: 6.087198198104765,

- ”mean”: 5.135097830934518,
- ”count”: 55.0

Absolute difference trimester 2:

- ”max”: 24.749099690650667,
- ”min”: 0.6480596018764686,
- ”std”: 3.5686772160242395,
- ”25pc”: 4.464146812138978,
- ”50pc”: 6.026485291046242,
- ”75pc”: 7.730095196420621,
- ”mean”: 6.64090360110811,
- ”count”: 233.0

Absolute difference trimester 3:

- ”max”: 27.589732124216823,
- ”min”: 0.6867050791128122,
- ”std”: 5.584556464260941,
- ”25pc”: 9.190157286174411,
- ”50pc”: 11.662282500140009,
- ”75pc”: 14.501014051515938,
- ”mean”: 12.041447934609026,
- ”count”: 47.0

Chapter 7

Conclusion

hahahahahah

References

[1] Luiz Camargo, Hegler Tissot, and Aurora Pozo. Use of backpropagation and differential evolution algorithms to training mlps. pages 78–86, 11 2012.

[2] A. Fitzgibbon, M. Pilu, and Robert B. Fisher. Direct least squares fitting of ellipses. *Proceedings of 13th International Conference on Pattern Recognition*, 1:253–257 vol.1, 1996.

[3] Ross Girshick. Fast r-cnn. 04 2015.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 11 2014.

[5] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP:1–1, 06 2018.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 06 2016.

[7] Thomas Heuvel, Dagmar de bruijn, Chris de Korte, and Bram Ginneken. Automated measurement of fetal head circumference using 2d ultrasound images. *PLOS ONE*, 13:e0200412, 08 2018.

[8] Salman Khan, Hossein Rahmani, Syed Shah, and Mohammed Bennamoun. A guide to convolutional neural networks for computer vision. *Synthesis Lectures on Computer Vision*, 8:1–207, 02 2018.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[11] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998.

[12] Jing Li, Yi Wang, Baiying Lei, Jie-Zhi Cheng, Jing Qin, Tianfu Wang, Shenli Li, and Dong Ni. Automatic fetal head circumference measurement in ultrasound using random forest and fast ellipse fitting. *IEEE Journal of Biomedical and Health Informatics*, PP:1–1, 05 2017.

[13] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. pages 3431–3440, 06 2015.

[14] Wei Lu, Jinglu Tan, and Randall Floyd. Fetal head detection and measurement in ultrasound images by an iterative randomized hough transform. *Ultrasound in medicine & biology*, 31:929–36, 08 2005.

[15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. pages 1–10, 01 2016.

[16] Frank Rosenblatt. *The Perceptron: A Probabilistic Model for Information Storage and Organization (1958)*, pages 183–190. 02 2021.

[17] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[19] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[20] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[21] Mark Villarino. Ramanujan's perimeter of an ellipse. 07 2005.

[22] Joseph Walsh, Niall O' Mahony, Sean Campbell, Anderson Carvalho, Lenka Krpalkova, Gustavo Velasco-Hernandez, Suman Harapanahalli, and Daniel Riordan. Deep learning vs. traditional computer vision. 04 2019.

[23] Rikiya Yamashita, Mizuho Nishio, Richard Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9, 06 2018.

[24] Zhao Yanling, Deng Bimin, and Wang Zhanrong. Analysis and study of perceptron to solve xor problem. pages 168 – 173, 12 2002.

[25] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2019. `http://www.d2l.ai`.