End-to-End Deep Image Matting Platform

END-TO-END DEEP IMAGE MATTING PLATFORM

BY

BOTAO XIAO, B.ASc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2019) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	End-to-End Deep Image Matting Platform
AUTHOR:	Botao Xiao B.ASc., University of Windsor, Windsor, Canada
SUPERVISOR:	Dr. Jun Chen
NUMBER OF PAGES:	x, 54

To Yijia Ren and Jinyu Xiao

Abstract

Image matting is a basic feature in most of image quality improvement applications and is considered as a fundamental problem in the computer vision field. In this paper, an End-to-End Image Matting Platform is proposed to segment the foreground and background by creating an alpha matte image.

My end-to-end image matting algorithm is a learning based network which contains two stages. The first stage is to create a tri-map image from an RGB image using segmentation neural network where tri-map images are used to locate the expected foreground objects with rough outlines. The second stage is an image matting neural network, and it takes the outputs from the first stage as prior knowledge to predict precise alpha matte images. With the help of image matting formula and the outputs from the second stage, contents in an RGB image can be easily split into foreground and background. I applied both training and evaluating on Adobe matting benchmark and Car Media 2.0's car oriented image matting dataset, and the outcomes demonstrated the convenience and superior performance of our algorithm compared to existing state of the art methods.

This paper put forward a web platform structure to integrate deep learning algorithms. I applied multiple strategies to enhance the performance of the platform. By using this platform, multiple users can work with different deep learning applications at the same time which dramatically increases the efficiency of the server usage.

Acknowledgements

First of all, I am grateful to my supervisor, Prof. Jun Chen, not only for his guidance using his knowledge but also his moral and emotional support in my daily life. His spirit of preciseness and consistent during exploring unknown area inspires me every single day throughout my Master program. Without his profound knowledge, this work would have not been successfully completed.

Secondly, I would like to show my thanks to my Colleagues Huan Liu, Chenxiao Niu and Zheng Liu for the suggestions and unfailing assistance. Collaboration with them is my great honor.

Furthermore, I want to show my respect to members in my defence committee, Dr. Sorina Dumitrescu and Dr. Jiankang Zhang, for their invaluable time and feedback.

Last but not least, I appreciate the companion of my wife Yijia Ren and my daughter Jinyu Xiao. Their silent dedication is my most precious treasure.

Thanks for all your encouragement!

Notation and abbreviations

CF	Closed-form Matting
CNN	Convolutional Neural Network
CPU	Center Processing Unit
CV	Computer Vision
EIMN	End-to-End Image Matting Network
FC	Fully Connected Layer
GPU	Graphics Processing Unit
MAP	Maximum A Posteriori
MSE	Mean Square Error
MVC	Model-View-Controller
ReLu	Rectified Linear unit
SSD	Solid State Drive

Contents

List of Figures

Chapter 1

Introduction and Problem Statement

1.1 Image Matting

Image Matting, known as the problem to estimate foreground and background objects in an RGB image, is widely applied in film editing and advertisement industries. With the restriction of time and space, human or objects cannot be placed at expected positions, so accurate estimations using matting algorithm may help us solve this problem. A lot of computer vision research communities, Chuang *et al.* (2001); Levin *et al.* (2007); Chen *et al.* (2013); Xu *et al.* (2017) to name a few, committed themselves to improving the performance due to the extensive uses of image matting algorithm.

Image matting is a pixel-level problem using an image I in RGB form as input. For i-th pixel in I, it is a composition of foreground pixel F_i and background pixel B_i (Fig. (1.1) and Eq. (1.1)),

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i \quad \alpha_i \in [0, 1]. \tag{1.1}$$

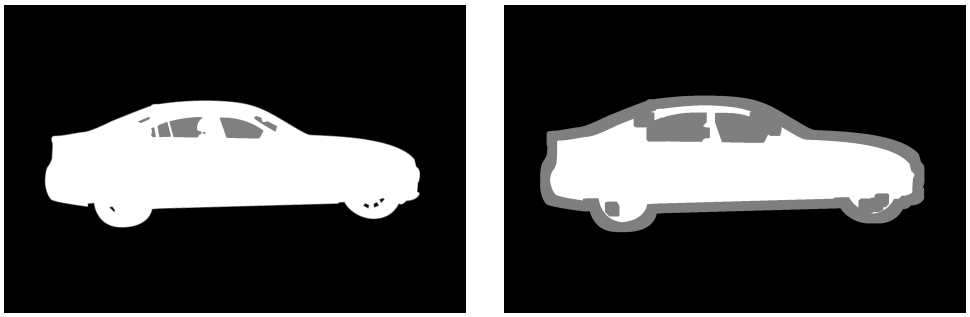
where α_i is the foreground capacity, also called alpha matte, for pixel i. The purpose of

Figure 1.1: Image composition equation using foreground, background and alpha matte. Explanation to Eq. 1.1

image matting is to estimate the α value for each pixel in the image.

Existing state of the art methods got amazing achievements in this fundamental CV area. Proposed approaches can be loosely concluded into three categories: 1) propagation-based method, 2) sampling-based method and 3) learning-based method. The propagation-based algorithms like Sun *et al.* (2004), Grady *et al.* (2005) and Levin *et al.* (2007), reformulated Equation (1.1) to create cost functions. Then by moving a sliding window from known regions (foreground and background) to unknown regions (partial unknown and completely unknown), their algorithms estimated the optimal alpha values by minimizing cost functions. Sampling-based methods also started from known regions to find the potential alpha values, then they applied different metrics to predict the alpha value for each pixel. Other than traditional methods, learning-based approaches like Long *et al.* (2015) and Xu *et al.* (2017) took the advantage of GPU-accelerated computing and used neural network to tackle the problem of alpha matte estimation.

Almost all of the proposed learning-based image matting methods have similar limitations. One of the limitations is about dataset quality because generating ground truth alpha matte for images requires high labor cost. Existing datasets (see e.g Xu *et al.* (2017); Rhemann *et al.* (2009)) manually created few ground truth alpha matte images and composed the RGB images by pasting foreground ground truth images onto different background images. Recently proposed deep learning methods trained using aforementioned dataset cannot achieve the same performance on real word images as they did not learn features



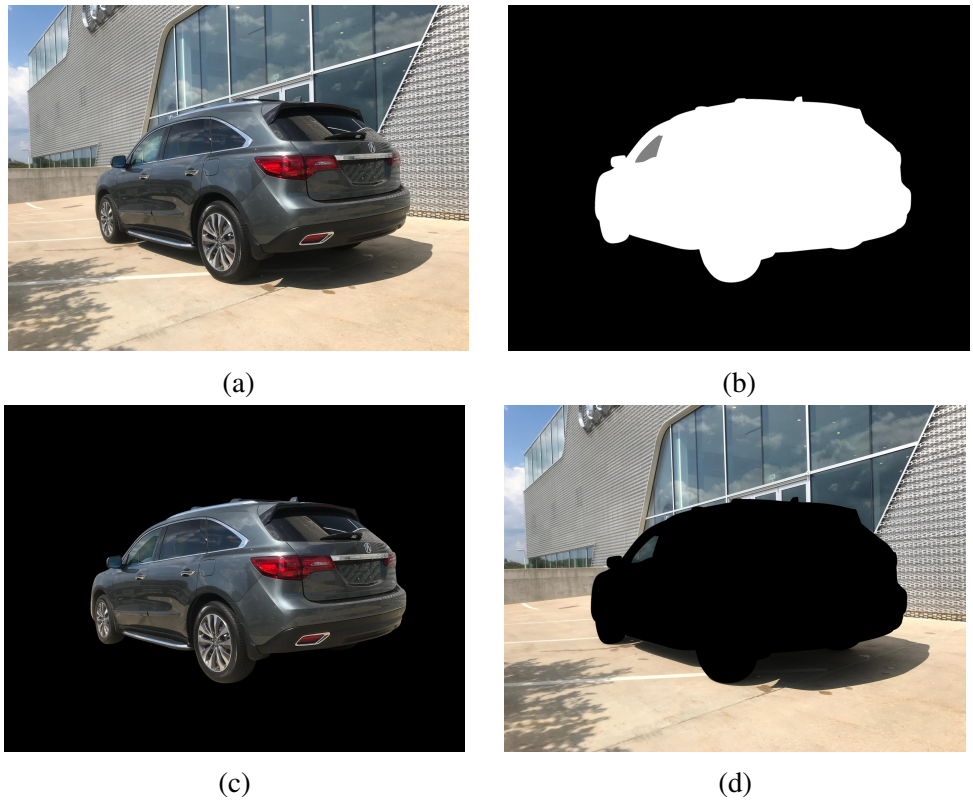
(a) ground truth (b) tri-map

Figure 1.2: Alpha matte (created by hand) and its corresponding tri-map image created with coarse index 36.

from real world images.

Another limitation is caused by the natural defect of image matting algorithms since they do not have the ability to figure out the dominant object in an image. Therefore, a rough hand drawn tri-map image (Fig. 1.2) is required as prior knowledge to image matting network. A tri-map image contains three regions, foreground, background and unknown regions. When facing high volume image processing requests, human work becomes a significant problem.

In this thesis, I would like to present some technical approaches to surmount the limitations. A car-oriented dataset with 28045 image sets was created. All images are car-oriented real word scenes and equipped with corresponding alpha mattes, foreground images, background images (Fig. 1.3). For removing the labor work on creating tri-map, I referenced and modified some learning-based segmentation methods (see e.g He *et al.* (2017); Long *et al.* (2015)) to create tri-maps automatically and engineering strategies were raised to integrate tri-map generation network and image matting network so it will be a completely end-to-end process.



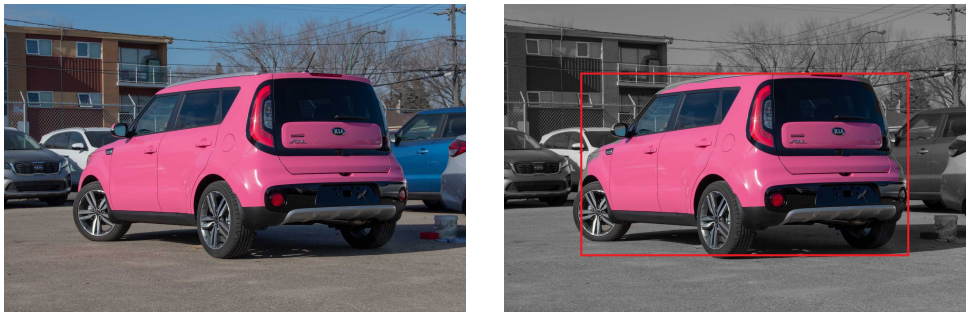
(a) (b)

(c) (d)

Figure 1.3: An image from our car-oriented dataset. (a) Real word car-oriented image. (b) Manually created alpha matte. (c) Foreground image. (d) Background image.

1.2 Tri-map Generation using Segmentation Algorithms

Segmentation is the algorithm which locates and catogorizes objects in an RGB image. We want to generate tri-map images by utilizing the functionality of segmentation networks. Our first shot is using semantic segmentation which is a pixel level classification algorithm aiming on categorizing all pixels in an image to different classes. As in a tri-map image, there are three classes which are foreground, background and unknown region. Then we changed our research direction to instance segmentation (Fig. 1.4) since we don't have to create the tri-map directly. Instance segmentation is a combination task of object classification and semantic segmentation in instance level. We can generate an approximate mask for the foreground object first and modify it to a tri-map image in the post processing.



(a) (b)

Figure 1.4: Image process using Instance Segmentation. (a) RGB Image. (b) Mask R-CNN result for (a): detect and color splash the foreground object.

1.3 Deep Learning Platform

Most of the deep learning algorithms require large computational power and memory which cannot be afforded by normal users. In this thesis, I will propose a web platform structure to deploy deep learning algorithms. I applied some cutting edge techniques to enhance the

performance of the platform and the details of this platform will be explained in Chapter 5.

1.4 Thesis Structure

Rest chapters of this thesis will be organized as follow: Chapter 2 illustrates previous work of image matting algorithms. Chapter 3 concludes my proposed End-to-end Image Matting Network (EDIMN) and explains the details of network fusion. Meanwhile, our car-oriented image matting dataset will be reviewed. Chapter 4 presents the implementation procedure of EDIMN using multiple GPUs and tensorflow framework. I also provide the results and performance comparison between EDIMN and other state of the art algorithms in this chapter. The deep learning platform will be introduced in Chapter 5 in the aspect of its structure and implementation. At the end, conclusion and discussions are listed in Chapter 6, talking about our advantages and related future work.

Chapter 2

Background and Related Work

In this chapter, I would like to review some of previous work related to my thesis. Over the last decade, remarkable contributions were put forward to tackle image matting and tri-map generation problems. Both topics can be categorized into traditional and learning-based methods, I will introduce some classical methods in this chapter.

2.1 Previous Work On Image Matting

Generally, there are two directions to solve image matting problem which are traditional methods (propagation-based and sampling-based algorithms) and learning based algorithms, as mentioned above in Chapter 1. Here I would like to review one algorithm for each of the three categories.

2.1.1 Propagation-based Methods

For propagation-based methods, such as Sun *et al.* (2004), Grady *et al.* (2005) and Levin *et al.* (2007), the first step is to reformulate e.q (1.1) into a way that they can estimate alpha

matte values in unknown regions from known regions (foreground and background region) by propagation.

Levin *et al.* (2007) proposed a propagation-based method called Closed Form Solution to Natural Image Matting (CF) to solve image matting problem. Their algorithm started demonstrating from gray scale images by reformulating Equation (1.1) to:

$$\alpha_i \approx a_i I_i + b_i \quad \forall i \in w. \tag{2.1}$$

where $a_i = \frac{1}{F_i - B_i}$ and $b_i = -\frac{B_i}{F_i - B_i}$ for i-th pixel in window w where α_i, a_i, b_i are unknown values. In order to distinguish known and unknown regions, CF requires a tri-map as prior knowledge, which is a manually created raw image to categorize all pixels in an RGB image into three types (foreground F, background B and unknown region U). CF propagates alpha values from known regions (F and B) to unknown regions U by moving a sliding window with size 3×3 to estimate the optimal alpha value in each sliding window. CF presented a cost function as:

$$J(\alpha, a, b) = \sum_{j \in I} \sum_{i \in w_j} (\alpha_i - a_j I_i - b_j)^2 + \varepsilon a_j^2 \tag{2.2}$$

where pixel j is the center of window w_j. In any small 3×3 sliding window, CF made an assumption that foreground F and background B pixels are smooth (a and b are constant), so α_j should have a optimal value and εa_j^2 is added to for numerical stability.

For an $N \times N$ window W_k, CF created two matrices G and $\bar{\alpha}$:

$$G = \begin{bmatrix} I_1 & 1 \\ I_2 & 1 \\ \dots & \\ I_{N^2} & 1 \\ \sqrt{\varepsilon} & 0 \end{bmatrix} \quad and \quad \bar{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_{N^2} \\ 0 \end{bmatrix} \tag{2.3}$$

Then they get:

$$G \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha} = \begin{bmatrix} I_1 a_k + b_k - \alpha_1 \\ I_2 a_k + b_k - \alpha_2 \\ \dots \\ I_{N^2} a_k + b_k - \alpha_{N^2} \\ \sqrt{\varepsilon} a_k \end{bmatrix} \tag{2.4}$$

where a_k and b_k presents the a and b in the sliding window w_k. With 2.4, CF reformulated E.q (2.2) to:

$$\sum_{j \in I} \sum_{i \in w_j} (\alpha_i - a_j I_i - b_j)^2 + \varepsilon a_j^2 \approx (G \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha})^T (G \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha}) = \parallel G \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha} \parallel^2 \tag{2.5}$$

Using 2.5, CF rewrited its cost function for each sliding window to:

$$J_k(\alpha, a, b) = \parallel G_k \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha_k} \parallel^2 . \tag{2.6}$$

Finding the optimal (α, a, b) can be divided into two steps which are: 1)Given α to find

optimal (a, b) pair (a^*, b^*) and 2) Finding optimal of $J(\alpha)$ with (a^*, b^*). Letting $\beta = \begin{bmatrix} a^k \\ b^k \end{bmatrix}$ and plug it into 2.5, CF got:

$$J_k(\bar{\alpha_k}) = (G_k \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha_k})^T (G_k \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha_k}) = (G_k\beta - \bar{\alpha_k})^T (G_k\beta - \bar{\alpha_k}) \tag{2.7}$$

So $[a^*, b^*]$ happens when $\frac{\partial J_k(\bar{\alpha_k})}{\partial \beta} = 2{G_k}^T G_k\beta - 2{G_k}^T\bar{\alpha_k} = 0$, so:

$$\beta^* = \begin{bmatrix} a^* \\ b^* \end{bmatrix} = ({G_k}^T G_k)^{-1} - 2{G_k}^T\bar{\alpha_k} \tag{2.8}$$

In 2.8, β was eliminated and optimal of $J(\bar{\alpha_k})$ relies only on $\bar{\alpha_k}$, also a and b in E.q (2.2) are removed. In a sliding window, CF assumed α for all pixels are the same (because all pixels in the sliding window are smooth), so $\alpha = \bar{\alpha_k}$. Then plug 2.8 into 2.6, they got:

$$J_k(\alpha, a, b) = \sum_{k\in I} \parallel G_k \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha_k} \parallel^2 = \sum_{k\in I} \parallel G_k(G_k^T G_k)^{-1} G_k^T \bar{\alpha_k} - \bar{\alpha_k} \parallel^2 \tag{2.9}$$

Let $\bar{G_k} = I - G_k(G_k^T G_k)^{-1} G_k^T$, CF got:

$$J_k(\alpha, a, b) = \sum_{k\in I} {\bar{\alpha_k}}^T {\bar{G_k}}^T \bar{G_k}\bar{\alpha_k} = \sum_{k\in I} {\alpha_k}^T L\alpha_k, \tag{2.10}$$

where L is Laplacian Matrix with equation:

$$L_{i,j} = \delta_{i,j} - \frac{1}{w_k}(1 + \frac{1}{\frac{\varepsilon}{|w_k|} + \delta_k^2}(I_i - \mu_k)(I_j - \mu_k)) \tag{2.11}$$

where μ_k is the mean pixel value in window, $|w_k| = N \times N$ (number of pixels in window), δ_k is the variance of pixel in current window and $\delta_{i,j}$ is the Kronecker delta.

Using E.q (2.10), this problem is changed to a Quadratically Constrained Quadratic Programs question with equation:

$$J_k(\alpha) = \alpha^T L \alpha + \lambda(\alpha^T - {b_s}^T) D_s(\alpha - b_s) \tag{2.12}$$

where D_s is a diagonal matrix with its diagonal mapping to each pixels in the window where pixels with known alpha values are mapped to 1 and unknown are mapped to 0 and b_s has the same shape as D_s with given alpha values for known pixels and 0 for unknown pixels. In order to find minimum of 2.12, CF took partial derivative to α and set it to 0:

$$\frac{J_k(\alpha)}{\alpha} = 2L\alpha + 2\lambda D_s(\alpha - b_s) = 0 \tag{2.13}$$

and the equation for α is:

$$\alpha = \lambda(L + \lambda D_s)^{-1} D_s b_s, \tag{2.14}$$

which is the optimal alpha value in the $N \times N$ sliding window.

It is obvious that CF method is one of the best traditional methods and will be compared with all of the later image matting methods.

2.1.2 Sampling-based Methods

Sampling-based methods also take tri-maps as prior knowledge. Statistical strategies are applied on known regions (foreground and background) to have a probability estimation on unknown regions. Berman *et al.* (2000) presented a Knockout algorithm as the estimation

method, Ruzon and Tomasi (2000) used Ruzon-Tomasi algorithm to extrapolate unknown region from determined foreground and background and Chuang *et al.* (2001) proposed a significant algorithm using bayesian approach.

Bayesian approach for Digital Image (BDI) raised by Chuang *et al.* (2001) tried to maximize the probability of $P(F, B, \alpha|I)$ using Bayesian equation and MAP technique. For a pixel in unknown region, we have one known value I but 3 unknown values: foreground F, Background B and alpha value α. So BDI can write the Baye's rule for this pixel:

$$P(F, B, \alpha|I) = P(I|F, B, \alpha)P(F)P(B)P(\alpha)/P(I). \tag{2.15}$$

In order to find the (F, B, α) values for holding max probability of $P(F, B, \alpha|I)$, BDI applies MAP algorithm and uses a sum of log likelihoods to re-write the e.q (2.15):

$$\begin{aligned} & argmax_{F,B,\alpha} P(F, B, \alpha|I) \\ & = argmax_{F,B,\alpha} P(I|F, B, \alpha)P(F)P(B)P(\alpha)/P(I) \\ & \approx argmax_{F,B,\alpha} L(I|F, B, \alpha) + L(F) + L(B) + L(\alpha), \end{aligned} \tag{2.16}$$

where $L(\cdot)$ is the log likelihood function $logP(\cdot)$. I is a known value so its probability is dropped. Now the question is changed to another question of find maximum $L(I|F, B, \alpha)$, $L(F)$, $L(B)$ and $L(\alpha)$. The first term can be presented by using e.q (1.1):

$$L(I|F, B, \alpha) = - \parallel I - \alpha F - (1 - \alpha)B \parallel /\sigma_I^2, \tag{2.17}$$

where DBI assumed I is in Gaussian distribution with expectation $\alpha F + (1 - \alpha)B$ and standard deviation σ_I. Here F, B and α are estimated values and they will lead to a estimated I ($I^{'}$). In order to have maximum probability of $P(I|F, B, \alpha)$ (or $L(I|F, B, \alpha)$),

BDI wanted $I^{'}$ to be close to I. BDI also suggested nearby N pixels as neighbours and created the Gaussian distribution with α_i^2. Then BDI used spatial Gaussian falloff g_i with $\sigma = 8$. Finally, they got the combined weight $w_i = \alpha_i^2 g_i$. For N neighbour pixels, BDI got mean and covariance matrix for F and B:

$$\bar{F} = \frac{1}{W}\sum_{i \in N} w_i F_i, \quad and \quad \bar{B} = \frac{1}{W}\sum_{i \in N} w_i B_i \tag{2.18}$$

and

$$\sum_F = \frac{1}{W}\sum_{i \in N} w_i (F_i - \bar{F_i})(F_i - \bar{F_i})^T \quad and \quad \sum_B = \frac{1}{W}\sum_{i \in N} w_i (B_i - \bar{B_i})(B_i - \bar{B_i})^T, \tag{2.19}$$

where $W = \sum_{i \in N} w_i$. With oriented elliptical Gaussian distribution, BDI presents:

$$L(F) = -(F - \bar{F})^T \sum_F^{-1} (F - \bar{F})/2. \tag{2.20}$$

For the $L(\alpha)$ term, BDI assumed nearby N neighbours shares the save α value so this term should be a constant. For solving the term $L(I|F, B, \alpha)$, BDI took partial derivative to e.q (2.16) on F and B and set it to 0 and got the following equation:

$$\begin{bmatrix} \sum_F^{-1} + C\alpha^2/\sigma_I^2 & C\alpha(1-\alpha)/\sigma_I^2 \\ C\alpha(1-\alpha)/\sigma_I^2 & \sum_B^{-1} + C(1-\alpha)^2/\sigma_I^2 \end{bmatrix} \begin{bmatrix} F \\ B \end{bmatrix} = \begin{bmatrix} \sum_F^{-1} \bar{F} + C\alpha/\sigma_I^2 \\ \sum_B^{-1} \bar{B} + C(1-\alpha)/\sigma_I^2 \end{bmatrix} \tag{2.21}$$

to solve optimal F and B, where C is an 3×3 identical matrix.

With optimal foreground F^* and background B^*, and assumed they are constant within nearby N neighbours, BDI got optimal α by taking partial derivative on α and set it to 0

and got the optimal α value with expression:

$$\alpha = \frac{(I - B) \cdot (F - B)}{\| F - B \|^2}. \tag{2.22}$$

Chuang *et al.* (2001) inspired later algorithms and got remarkable performance. Unfortunately, sampling-based methods requires significant calculation power and its accuracy drops when images are too complicated. Meanwhile, it requires tri-map as prior knowledge to complete the calculation.

2.1.3 Learning-based Methods

With the improvement of parallel computing units, the speed of computer is faster than it was in last decade and it leads to the fast development on learning-based methods, in which deep learning with convolutional neural network is one of the most important ones. Xu *et al.* (2017) proposed a learning-based algorithm Deep Image Matting (DIM) (Fig. 2.1). The network structure was composed by two parts: 1) A deep convolutional encoder-decoder net and 2) a refine net. Their method came first on alphamatting.com benchmark (Rhemann *et al.* (2009)) in terms of SAD metric and ranked second on comparison of MSE and Gradient metric.

Encoder-Decoder Net

The first stage of DIM is an encoder-decoder structure. Input to this stage is a 4-channel tensor where the first three channels are the three color channels of an RGB image. The last channel is the tri-map image of the corresponding RGB image. Xu *et al.* (2017) (Fig.

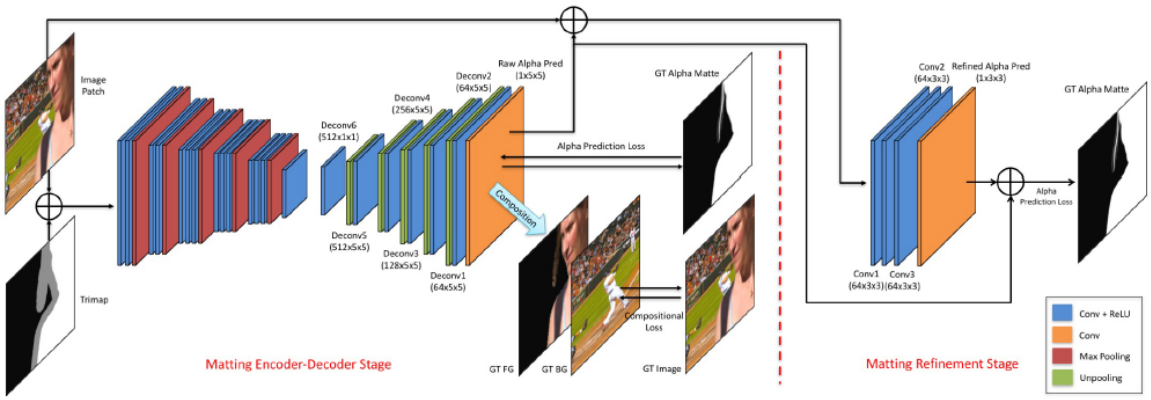


Figure 2.1: Network structure of Deep Image Matting Xu *et al.* (2017), where the network is composed of encoder-decoder stage and a refine stage.

2.2) modified VGG16 (Simonyan and Zisserman (2014)) and used it as the encoder network, which down-samples images and extracts features from complex contents by using strategies like convolution, padding, pooling, dropout and flatten. Here I would like to give a small introduction to these techniques.

- Convolutional Layer: Convolution is a process using a small designed size matrix (called kernel) to traverse the whole images with a fixed step size. For each step, the output is the production between the input tensor and the kernel. The process of convolution extracts feature map and the purpose of training is to update the values saved in the kernel.

- Padding Layer: Since convolution process over a image may reduce the size of the input matrix, padding strategies are applied to maintain the output size to make it in the same as the input I. There are several options for padding and VGG16's strategy is padding 0 at edges.

- Pooling Layer: Redundant features may be extracted during convolution process so pooling layers drop some of the features to reduce the calculation load by taking

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 (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
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**
maxpool					
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**
maxpool					
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**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.2: VGG16 Network Structure

maximum or average within a pool.

- Dropout Layer: Dropout is a strategy during training process and it eliminates the over-fitting possibility.

- Fully Connected Layer: Fully Connected Layer is normally used in classification problems. It parses features and maps objects to their corresponding class.

DIM expended the first convolutional kernel in VGG16 from $[3, 3, 3, 64]$ to $[3, 3, 4, 64]$ in form of $[k, k, i, o]$ where k is kernel size, i and o are input and output channel numbers

to fit their special four-channel input. Image matting does not require object classification ability, so FC was reshaped to a $[3, 3, 512, 4096]$ convolutional layer. The decoder part in DIM used deconvolutional layers to restore down-sampled images to desired rough alpha matte images α_p.

Loss function for this stage is composed by two parts (alpha loss and compositional loss):

$$L_{total} = w_\alpha L_\alpha + (1 - w_\alpha)L_c, w_\alpha \in [0, 1], \tag{2.23}$$

where w_α is the weight of alpha loss.

In image matting algorithm, there is a trick that we only calculate loss and apply back propagation on unknown regions. The reason is tri-map has provided known regions to the network so the prediction on that is always accurate and not necessary to be updated. Alpha loss L_α is defined as the root mean squared error on unknown regions, here N is the number of pixels in the unknown region:

$$L_\alpha = \sqrt{\frac{1}{N}\sum_i (\alpha_p^i - \alpha_g^i)^2 + \epsilon^2}, \tag{2.24}$$

to compare predicted and ground truth alpha matte (α_p and α_g) and L_c is defined as:

$$L_c = \sqrt{\frac{1}{N}\sum_i (c_p^i - c_g^i)^2 + \epsilon^2} \tag{2.25}$$

where c_p is created by predicted alpha matte α_p, foreground F, background B using e.q (1.1). Same as alpha loss, compositional loss is only applied on unknown regions.

Table 2.1: Refine net

Input ($224 \times 224 \times 4$)
Conv3-64 [3, 3, 4, 64]
Conv3-64 [3, 3, 64, 64]
Conv3-64 [3, 3, 64, 64]
Conv3-64 [3, 3, 64, 1]

Refine Net

The refine net contains four convolutional layers which is designed to fix small errors in α_p. Input to Refine stage is a concentration of an RGB image c_g and the result from the encoder-decoder stage (α_p), and output (α_{diff}) is a estimated difference between α_p and ground truth alpha matte. So the final result (α_{final}) is the sum of the result from refine net and α_p:

$$\alpha_p = \alpha_{raw} + \alpha_{diff}. \tag{2.26}$$

Only alpha loss L_α is applied here to update the refine net with equation:

$$L_{refine} = \sqrt{\frac{1}{N}\sum_i (\alpha^i_{final} - \alpha^i_g)^2 + \epsilon^2}. \tag{2.27}$$

2.2 Previous work On Tri-map Generation

Semantic segmentation is a problem aiming on pixel-level classification. In this problem, semantic segmentation algorithms assign labels to all pixels in an image. When we rethink the problem of tri-map generation, we found it could be changed to a semantic segmentation problem, in which all pixels need to be categorized into three classes: foreground, background and unknown, denoted as F, B, U. By training the network with ground truth tri-map, we want the network labeling all pixels in an RGB image into mentioned three

classes.

Before the population of learning-based semantic segmentation methods, multiple traditional methods are proposed to solve this difficult problem. Kohler (1981) proposed a method based on thresholding, Shi and Malik (2000) and Felzenszwalb and Huttenlocher (2004) solved this problem using graph partitioning methods and Sathya and Manavalan (2011) presented a clustering-based algorithm. Long *et al.* (2015) proposed a learning-based end-to-end semantic segmentation algorithm called Fully Convolution Network (FCN) and a deep convolutional encoder-decoder (SegNet) from Badrinarayanan *et al.* (2017) improved their performance. Mask-RCNN from He *et al.* (2017) realized both semantic segmentation and instance classification in their instance segmentation network, which can be used to generate tri-map images indirectly.

2.2.1 Learning-Based Methods

Long *et al.* (2015) firstly proposed a network called Fully Convolutional Network (FCN, Fig. 2.3) to tackle this problem. Before FCN, CNN is majorly used to solve object classification problems so the last several layers were fully connected layer. FCN updated FC to convolutional layers and changed the task from object-level to pixel-level classification. He *et al.* (2017) proposed an convolutional neural network Mask R-CNN which can locate the object and apply semantic segmentation within the bounding box of each instance. We tried both methods on tri-map generation in our research and achieved great success. In next several paragraphs, I will explain the functionality of both methods.

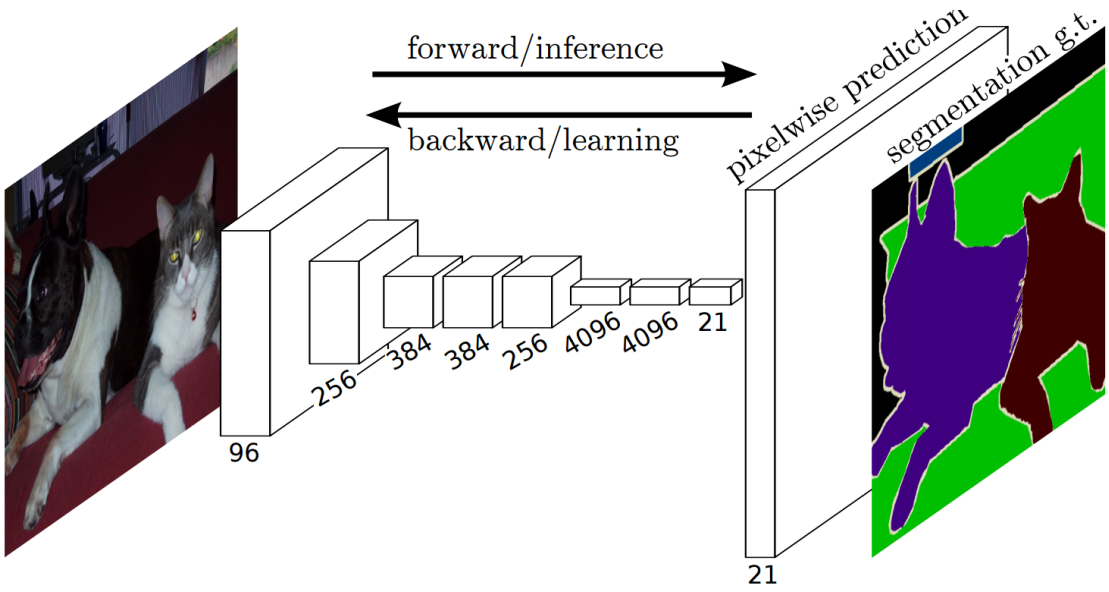


Figure 2.3: Fully Convolutional Network. take from Long *et al.* (2015)

Fully Convolutional Network

The most significant improvement from FCN is they converted Fully connected layer to an convonlutional layer. Fully connected layer takes the feature map as input and creates a $1 \times 1 \times N$ matrix where N is number of classes. Values saved in the result present the probability of current image matching all classes. Converting FC to convolutional layer has the physical meaning that we change the size of the filter to the size of the input tensor, so it will apply pixel-level classification to an image. Finally, output from the converted convolutional layer is no longer a feature map but a heat map (see figure 2.4) and each pixel in the heat map saves the information of a region in the input image.

Pixel wise Estimation

Being processed by the convolutional layers, image features are reserved by a heat map and the information is not human readable. So FCN used up-sampling and deconvolutional

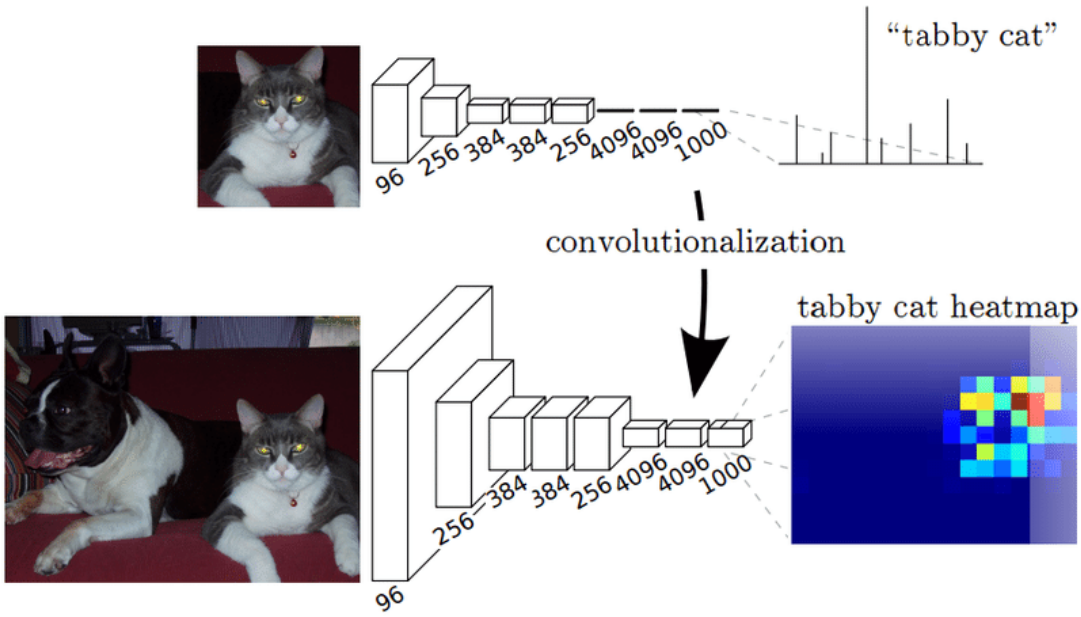


Figure 2.4: Change Fully Connected Layer to convolutional layer. Taken from Long *et al.* (2015)

layers in their last stage to restore the down-sampled heat map back to the input image size and each pixel in the output indicates the category it belongs to.

Skip Net

Convolutional layers extract the image features and create a heat map, whose size is $\frac{1}{32}$ of the input image size after processing by five convolutional layers which will cause some information loss. So directly applying restoration operation on the heat map may not get a precise outcome. In order to solve this problem, FCN proposed the method skip net (Fig. 2.5). As conv3 and conv4 layers saved a feature map whose sizes are $\frac{1}{8}$ and $\frac{1}{16}$ of the input image, skip net fuses the information from different layers and refines a more precise output.

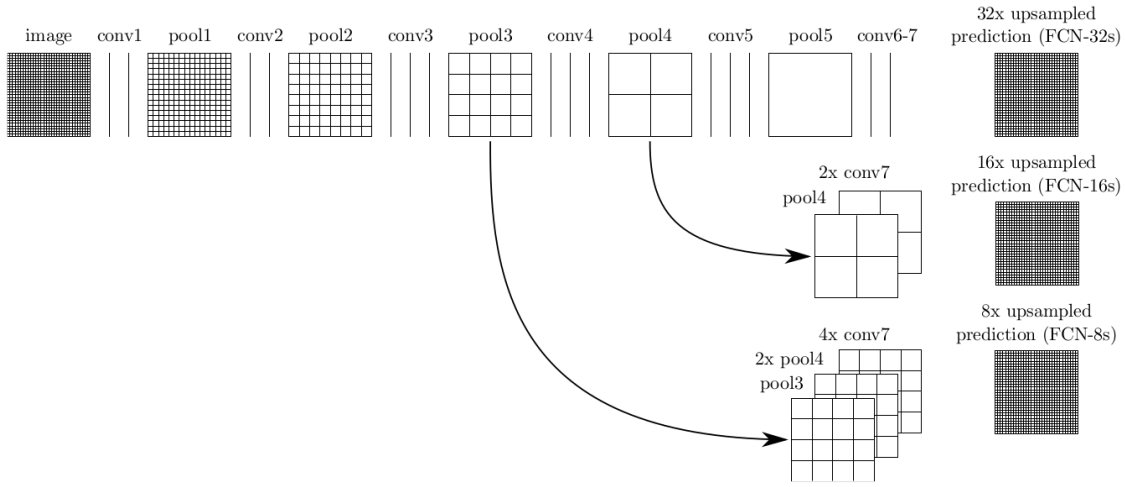


Figure 2.5: FCN skip net model. taken from Long *et al.* (2015)

2.2.2 Mask-RCNN

Fully convolutional Network (FCN) almost solved the tri-map generation problem except for one weakness: during pixel level prediction, FCN cannot distinguish the foreground and background objects if they are in the same category, which requires the network to have the ability of instance segmentation. Based on the Fast R-CNN from Girshick (2015), He *et al.* (2017) proposed a learning based method Mask R-CNN (Fig. 2.6) to solve the problem of instance segmentation. Mask R-CNN not only realizes semantic segmentation functions but also casts all pixels from one instance in to a group.

As a developed version of Fast R-CNN, results from Mask R-CNN are composed by three parts: bounding boxes which locate the position of objects, classification results showing the class of each bounding box and a mask for each candidate object. The first stage of Mask R-CNN is a Feature Pyramid Network (FPN), it is constructed by a backbone network which creates multiple feature map representations for an input image. For each point in a feature map, Mask R-CNN creates a Region of Interests (RoI) which may

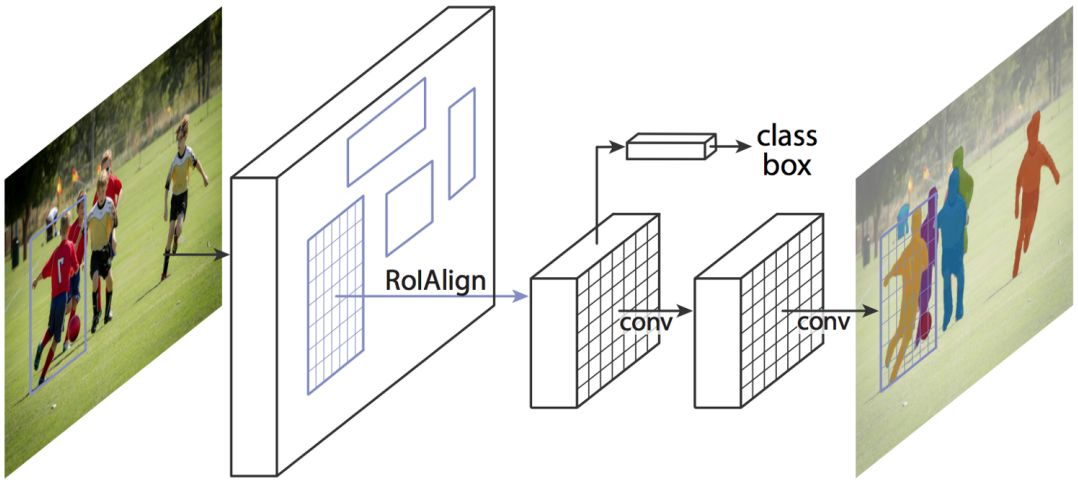


Figure 2.6: Mask R-CNN model. taken from He *et al.* (2017)

or may not contains expected objects. Then all candidate RoI regions are fed to Region Proposal Network (RPN) where incorrect candidate RoIs are filtered. Feature map is a down-sampled version of input images (not in the same size) and not human readable, Mask R-CNN proposed an RoIAlign operation to match each pixel in a feature map to the pixels in input image and RoIAlign significantly increase the accuracy of mask generation. The last stage of Mash R-CNN is constructed by two branches, one uses Fully Convolutional Network (FCN) in each RoI to create a mask and the other branch passes the RoI patterns to Fully Connected Layers for classification and bounding box regression. The loss function for Mask R-CNN is presented as:

$$L_{total} = L_{cls} + L_{box} + L_{mask}, \tag{2.28}$$

where L_{cls} is classification loss, L_{box} is the loss for bounding box and L_{mask} means the mask loss.

Mask R-CNN achieved a better performance compared with other state of the art algorithms, and even on the images which contain overlaps between same class's instance, Mask R-CNN can clearly figure out the boundaries and it is the basic requirement to tri-map generation.

Chapter 3

The Proposed Algorithm

In this chapter, I will propose a fully automatic image matting algorithm called End-to-End Deep Image Matting (EDIM). Our image matting network contains two stages which implement the function of tri-map generation and image matting. We also create our own car-oriented image matting dataset to eliminate the weakness of current existing datasets and the following subsections in this chapter will have more discussion about our car-oriented dataset, trimap generation algorithm and image matting methods. The results of our algorithm will be demonstrated in Chapter 4.

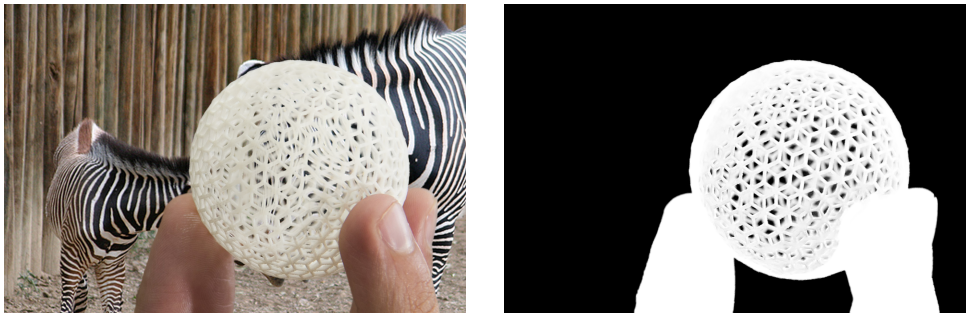
3.1 New Car-oriented Dataset

Rhemann *et al.* (2009) proposed a matting benchmark on alphamatting.com (Fig. 3.1). It has only 27 image sets and 8 test image sets where each image set contains an RGB image and its corresponding alpha matte. This benchmark achieved a great success since it was the first benckmark and accelerated the speed of training process for learning-based methods. However, restricted by its low volume and diversity, designed networks are easy

(a) RGB Image (b) Alpha Matte Image

Figure 3.1: Example from alphamatte.com: (a) RGB Image. (b) Alpha matte of (a)

.



(a) RGB Image (b) Alpha Matte Image

Figure 3.2: Example from Adobe dataset: (a) RGB Image. (b) Alpha matte of (a)

.

to over-fit.

Xu *et al.* (2017) proposed a new image matting dataset collaborating with Adobe (Fig. 3.2). This dataset contains both training and evaluating sets and training set has 493 foreground objects with their corresponding alpha matte images. By pasting each foreground image to N new background images, they create a training set with 49300 ($N = 100$) images. They also used the same method to create an evaluation set which contains 50 foreground objects and totally 1000 ($N = 20$) image sets. This large dataset is a tremendous

success since it enlarges the image matting dataset and covers more complex objects including complex and semi-transparency objects. However, networks trained on this dataset will not achieve good performance on real world images because the composed images contain bias at the intersections between foreground object and background so network will learn different keys which cannot be applied on real world images directly.

In order to solve the problem caused by composed images, we created our own car-oriented image matting dataset. Collaborating with Car Media 2.0, we created a dataset (Fig. 3.3) with 28045 images sets and none of the images are synthetic. This dataset contains images including multiple types of vehicles like sedan, truck and van. We took photos for cars in different scenes (e.g parking lot, factory, store etc) and manually created alpha matte images. Compared with alphamatting.com and Adobe dataset, all images in our dataset are from real world. Meanwhile, our dataset has more critical scenes like background car is overlapping with the foreground car and some them even share the same color. Image matting networks trained by our dataset will learn more features from real world than alphamatting.com and Adobe dataset and the complex context is a huge challenge to existing methods.

3.2 End-to-End Deep Image Network

3.2.1 Tri-map Generation Network

Regarding to the limitation on manual work of tri-map generation mentioned in Chapter 1, we proposed an algorithm for tri-map generation. This problem can be solved by 2 ways: 1) Use semantic segmentation to predict tri-map directly or 2) Use instance segmentation to create mask for each objects then create trimap using the mask by post processing.

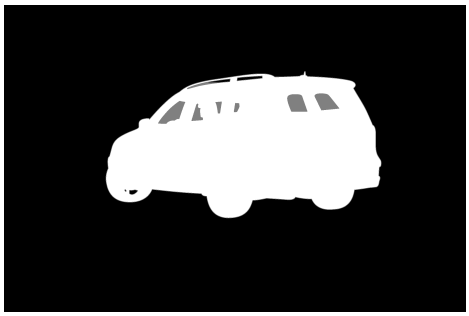
(a) RGB Image

(b) Alpha Matte Image

(c) Foreground Image

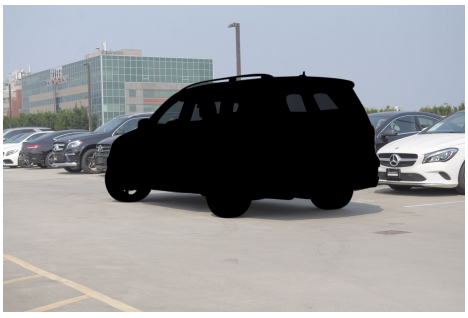
(d) Background Image

(e) RGB Image



(f) Alpha Matte Image

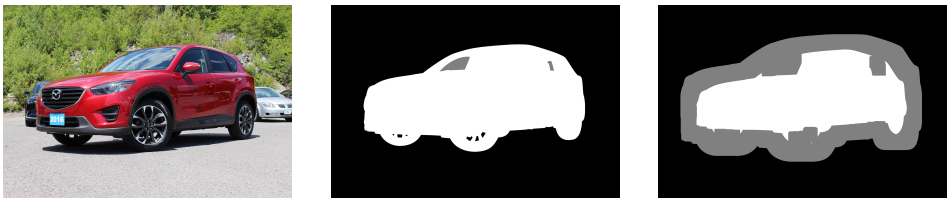
(g) Foreground Image



(h) Background Image

Figure 3.3: An image from our Car-oriented dataset.

.



(a) (b) (c)

Figure 3.4: Inputs to Trimap Generation Network. (a) Original Image. (b) Alpha matte to create ideal trimap. (c) Ideal trimap.

.

The first options is creating tri-map by semantic sengmentation. There are three regions in an ideal tri-map image: Foreground (F), Background (B) and Unknown Region (U). We can consider these three regions as three different classes. By pixel level estimation, semantic segmentation network tries to categorize each pixel in the graph into one of the three classes. We trained and evaluated the network FCN-8s Long *et al.* (2015) using our dataset, which is a basic fully convolutional encoder network.

- Input: An RGB image is fed to the network as input and an ideal tri-map corresponding to the RGB image is also provided as ground truth during the training process. As the alpha matte for RGB image is given, we can create the tri-map automatically by grey dilation with a coarse index. We also decide the dilation coefficient, which determines how much we want the critical unknown area to expand between foreground and background. The unknown areas expand in both directions and completely cover all intersections foreground and background. Also semi-transparent regions in alpha matte images will be marked as unknown (Fig. 3.4) with some expansion because the edge of unknown region also need to be determined.

- Network: We referenced network structure from FCN-8S.

- Loss function: We use RMSE (root mean squared error) as the loss function:

$$L_{fcn} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{T}_p - T_{gt})^2}, \tag{3.1}$$

where T_p is the predict trimap and T_{gt} is ground truth trimap.

Semantic segmentation performs well on most of the images except a condition that foreground and background instance is in the same the class and they have overlap. In order to solve this problem, we changed our research direction to instance segmentation using Mask R-CNN (Fig. 3.5). We create rough masks, as well as class labels, for each object in the RGB images and apply post processing over the predict masks. Since we followed the network structure and loss function from Mask R-CNN, here I only introduce our designed input and post-process.

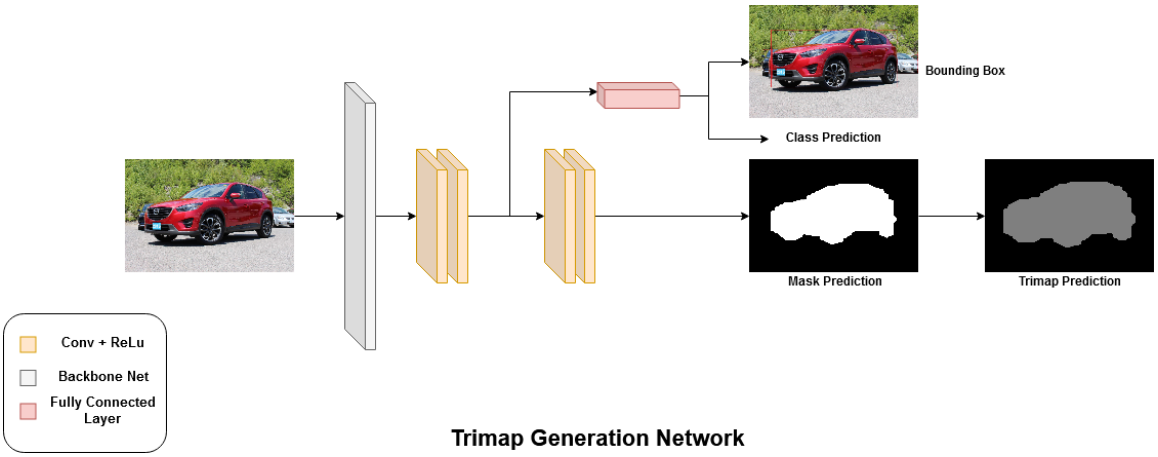


Figure 3.5: Tri-map Generation Network

- Input: Mask R-CNN uses RGB images and their labeling files in form of VOC annotation format, where all masks are created by polygons. We modified the code to make Mask R-CNN take RGB image as input to the network and their alpha matte

images as mask ground truth, which is more precise than the polygon shapes labeled by VOC annotation.

- Post-processing: In order to find the dominant instance in an image, we made an assumption that its mask contains the most pixels. We create a priority queue for each of the masks which will sort the predicted mask according to their sum values (adding all pixel values in the mask together). The time complexity of this operation is $O(N * n \log n)$ where N is total instance number and n is the number of instance save in current class. We take the largest mask from the ideal class as the candidate mask for foreground instance. Then we apply grey dilation to the candidate mask and use it as a tri-map image. In our method, we changed all foreground regions to unknown since instance segmentation does not have the function of unknown region estimation and semi-transparent regions will be predicted as foreground.

3.2.2 Modification to Deep Image Matting

The process of solving image matting problem is to solve equation (1.1) with given parameter I_i for i-th pixel in the image, and the equation cannot be directly solved using mathematical techniques. Convolutional neural network (CNN), which can be considered as a complex mathematical model, can extract information from images in a higher level. We modified the encoder-decoder model from Xu *et al.* (2017) and trained this model as our image matting network (Fig. 3.6).

- Network Structure: Our network structure is very similar to Deep Image Matting but with some modifications. It is also in encoder-decoder structure, where the encoder is a modification to VGG16 (Fig. 2.2). The first convolutional layer is modified to

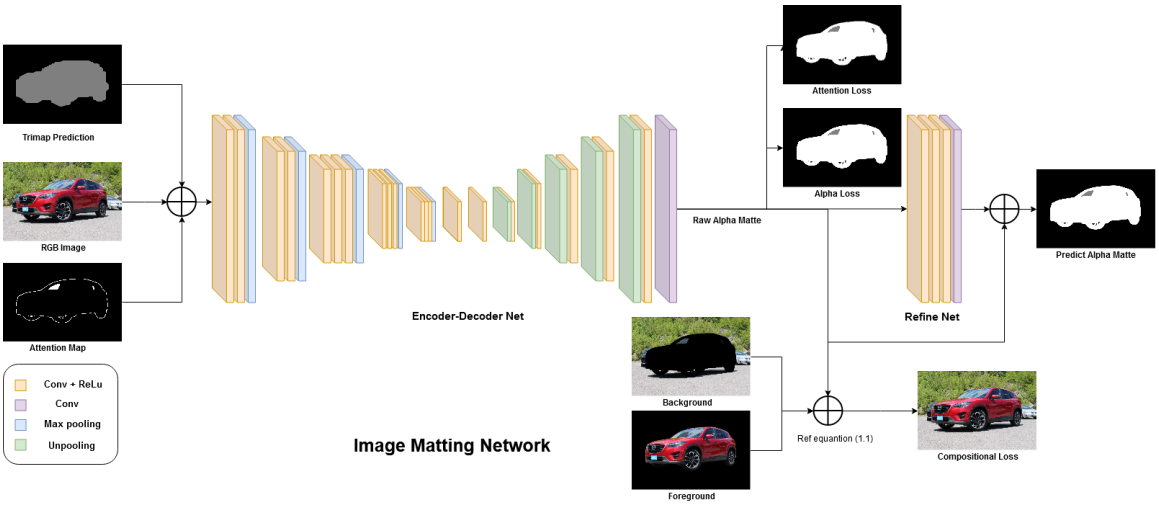


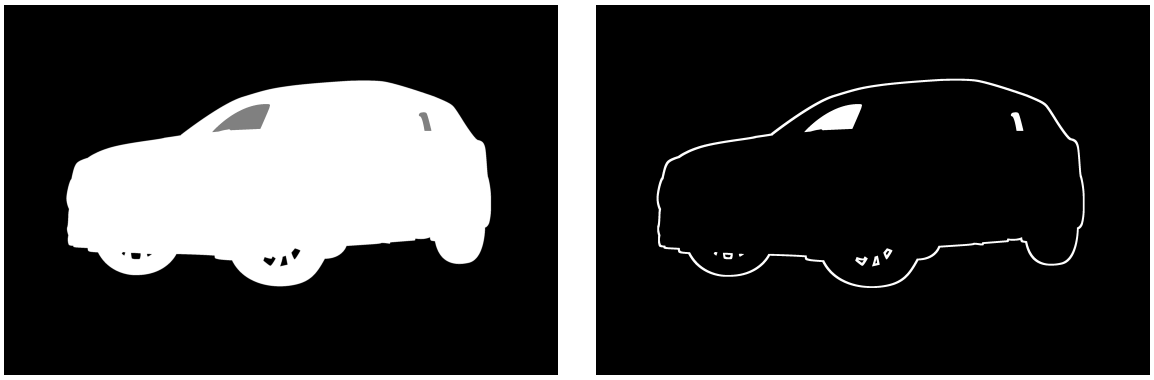
Figure 3.6: Deep Image Matting Network

4 channels where the first three channels are the RGB image and the fourth channel is the estimated tri-map. Encoder structure helps us extract the high level context from the image and tri-map is for locating the dominant object we want to extract from the image and define the regions for loss calculation. The attention map marks important regions the network needs to pay more attention to by applying a new loss called attention loss. After the processing of encoder, input tensor is down sampled by convolutional layers and features in the image are extracted. In order to predict an alpha matte image in the same size as the input, we applied up-sampling strategies. Different from Deep image matting, which uses transpose convolutional layers (aka deconvolutional layer) to restore low resolution features to an alpha matte, we replaced transpose convolutional layers by maximum unpooling layers. Both layers (transpose convolutional and unpooling) are restoration process to upsample low resolution image back to high resolution, and the difference between them is transpose convolutional layers store parameters and those parameters can be updated by back propagation but unpooling is a copying algorithm which fills the neighbours

of a pixel with a value. The backward propagation for training is both time and calculation power consuming especially on our complex car-oriented dataset, so we replaced all transpose convolutional layers with unpooling to speed up the training process. The last stage (Refine Net) uses three convolutional layers to predict the difference between the ground truth and the output from encoder-decoder structure and the final result is the sum of the raw alpha matte from encoder-decoder structure and the estimated difference. By adding the refine net, outputs become sharper and more precise.

- Input: The training inputs to the image matting network are RGB image O, its corresponding tri-map T created by tri-map generation net, ground truth alpha matte and an attention map M_a, foreground image F and background image B. The direct input to the network is a four channel tensor where the first three channels are the three channels in the RGB image and the last channel is the predicted tri-map. The attention map is specially designed for attention loss calculation. The attention map marks the intersection regions between foreground and background as a precise boundary is the basic requirement to image matting network. In the attention map (Fig. 3.7), attention areas will be marked as 1 and other pixels are 0.

- Loss Function: There are three terms in our encoder-decoder network's loss function which are alpha loss, attention loss and compositional loss. Alpha loss is the root mean squared error (rmse) between the ground truth alpha matte and our prediction (raw alpha matte). Our network uses the predicted alpha matte, foreground and background image to compose an RGB image using equation (1.1) and the compositional loss is rmse between the composed image and input RGB image. The attention loss makes the network pay more attention to boundary check as it offers additional



(a) (b)

Figure 3.7: Attention Map from Alpha Matte Image. (a) Attention Map. (b) Corresponding Alpha Matte.

.

error on edge areas and the model will use this error to update the model by back propagation.

$$L_{total} = (w_\alpha)L_\alpha + (w_a)L_a + (1 - w_\alpha - w_a)L_c, w_\alpha, w_a \in [0, 1], \tag{3.2}$$

where L_α is the alpha loss (e.q 2.24) and L_c is compositional loss (e.q 2.25). L_a is the attention loss calculated based on attention map $M_{attention}$ aiming on handling complex intersections between foreground and background objects and equation is:

$$L_a = \sqrt{\frac{1}{M}\sum_i((\alpha_p^i - \alpha_{gt}^i) \times M_{attention}^i)^2 + \epsilon^2}, \tag{3.3}$$

and that loss calculates the rmse on attention regions. The purpose of the refine net is to increase the accuracy of the raw alpha matte created by the encoder-decoder structure and the loss function is the root mean squared error between the final predicted

alpha matte and the ground truth:

$$L_{refine} = \sqrt{\frac{1}{N}\sum_{i}(\alpha^{i}_{final} - \alpha^{i}_{gt})^2 + \epsilon^2}, \tag{3.4}$$

where the final alpha value on i-th is the sum of the prediction from encoder-decoder structure and correction from refine net with representation:

$$\alpha^{i}_{final} = \alpha^{i}_{p} + \alpha^{i}_{refine}. \tag{3.5}$$

3.2.3 End-to-End Deep Image Matting

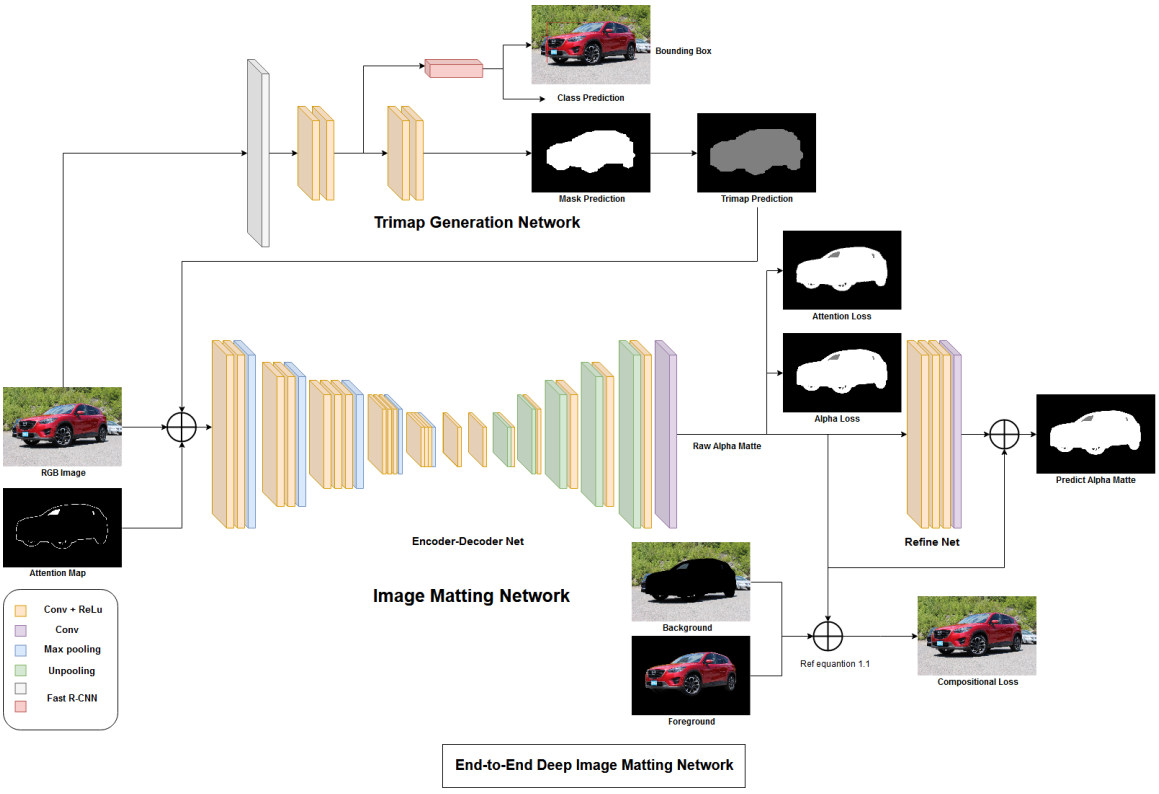


Figure 3.8: End-to-End Deep Image Matting Network

Our End-to-End Deep Image Matting (EDIM) (see figure 3.8) is a combination of Tri-map Generation Network (TGN) and Deep Image Matting (DIM) network. The generated tri-map from the first stage is the fourth channel of the image matting network's input. During the image matting process, tri-map is automatically generated and the result from our network will be shown in next chapter.

Chapter 4

Implementation and Experimental Result

This chapter introduces the training and testing details and lists the results from our proposed method as well as the comparison between different state-of-the-art methods.

4.1 Dataset Training and Testing

We used Adobe dataset Xu *et al.* (2017) as the first training dataset since this dataset contains more classes of objects and pre-training on this dataset allows our model save more features. This dataset contains 493 unique objects and by pasting each of the foreground objects to 100 different backgrounds, there are 49300 image sets in the dataset (including RGB image and their corresponding alpha matte images). It is also an important benchmark to check the functionality if image matting network. The comparison between our network with other state-of-the-art algorithms are carried out at this point.

We tried to apply the trained model using Adobe dataset to real world images, but the

results are not as good so we fine tuned the pre-trained model on our car-oriented dataset (see Fig. 3.1). There are totally 28045 image pairs in our dataset and we randomly took 45 image pairs from the dataset to evaluate the performance of our network to make sure the validation sets were invisible to the network during training process. Finally we have 28000 image sets for training and 45 for evaluation from our car-oriented dataset.

4.2 Training Details

Training of convolutional neural network is tensor computation intensive so only GPU-equipped work station can meet this requirement. Our deep learning work station has 4 NVIDIA GTX 1080ti GPUs with 11G memory. CPU in our work station is i7 7700K CPU with 12 logic cores and RAM and SSD storage are 64G and 2T. Our program is implemented on Google's machine learning platform Tensorflow Abadi *et al.* (2015) which provides GPU acceleration apis for training and evaluation on both single and multiple GPU environments.

Our proposed End-to-End Deep Image Matting network contains two parts which are Tri-map Generation Network (TGN) and Deep Image Matting Network (DIM) and these two parts are trained separately. The first stage reused the network structure of Mask R-CNN He *et al.* (2017) with some modifications, the size of input image is not restricted, with the consideration of GPU memory and the performance requirement, we decided to feed 1 image set to each GPU with no crop or reshape. We downloaded a pre-trained Mask R-CNN model and fine tuned it on our car-oriented dataset. The mask loss dropped from 0.096 to 0.001 after 20 epoches, which is considered as a good stop point. The second stage (EDIM) is an encoder-decoder model, and input images are processed by 5 max-pool layers and 5 unpooling layers, so the size of input is required to be multiple of $2^5 = 32$ otherwise

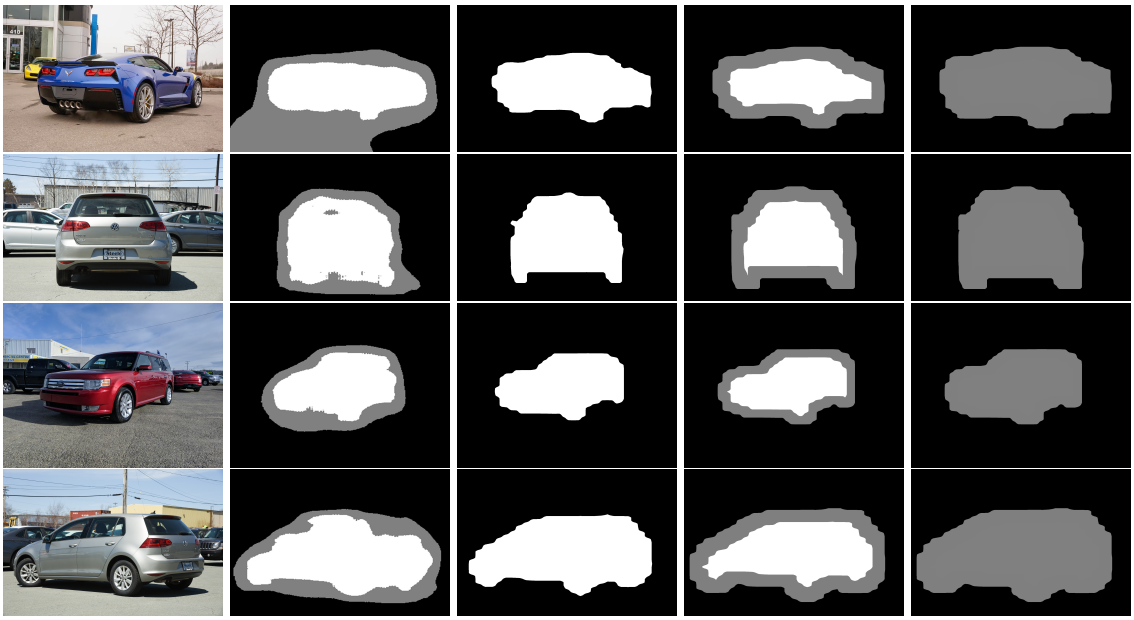
it won't be successfully restored by unpooling layers. In order to fit the video memory and have a larger batch size, the input image size is determined to be 320×320 and batch size is 12 for each GPU. All images are serialized into tfrecord format and shuffled before feeding to the network. We randomly select sizes from 320×320, 480×480 and 640×640 to crop image patterns and resized them to 320×320 since this action will provide the network an ability to understand the image in different resolutions. The training loss drops from 0.465 to 0.003 after 47 epoches on Adobe dataset. Then we fine tuned the pre-trained model to our car-oriented dataset (see 3.1) and loss dropped from 0.210 to 0.003 in 100 epochs.

4.3 Evaluation Details

Evaluation process is similar to training process but the only difference is that back-propagation is not applied. As mentioned in Section 4.2, our network requires input images' size (both height and width) to be a multiple of 32 and our development platform Tensorflow uses static graphs to construct the network which means once the static graph is created, the tensor size in all layers should be fixed so we must have a fixed input size at the beginning. If an image's size doesn't match the requirement, we will resize it to the closest number which is a multiple of 32 to feed the network so it won't lose too much information during this process, and the last stage uses bicubic algorithm to resize the predicted alpha matte image back to the size of the input image.

4.4 Tri-map Generation Results

As mentioned in Chapter 3, instance segmentation has a better performance on tri-map predicting than semantic segmentation. From Fig. 4.1, we can clearly see that the masks from Mask R-CNN precisely match the shape of foreground object while semantic segmentation only predict a pixel-level classification map where the shape of instance is not considered. As a conclusion, tri-map images created by Mask R-CNN can provide more prior knowledge to image matting network than those created by semantic segmentation though we still consider semantic segmentation for creating tri-map as a good option.



(a) (b) (c) (d) (e)

Figure 4.1: Generated Tri-map Comparison between Semantic Segmentation and Instance Segmentation. (a)RGB Image. (b)Predicted Trimap using FCN. (c)Mask from Mask R-CNN. (d) Intermediate tri-map generated by (c)

. (e) Tri-map feed to image matting network.

4.5 Deep Image Matting Results

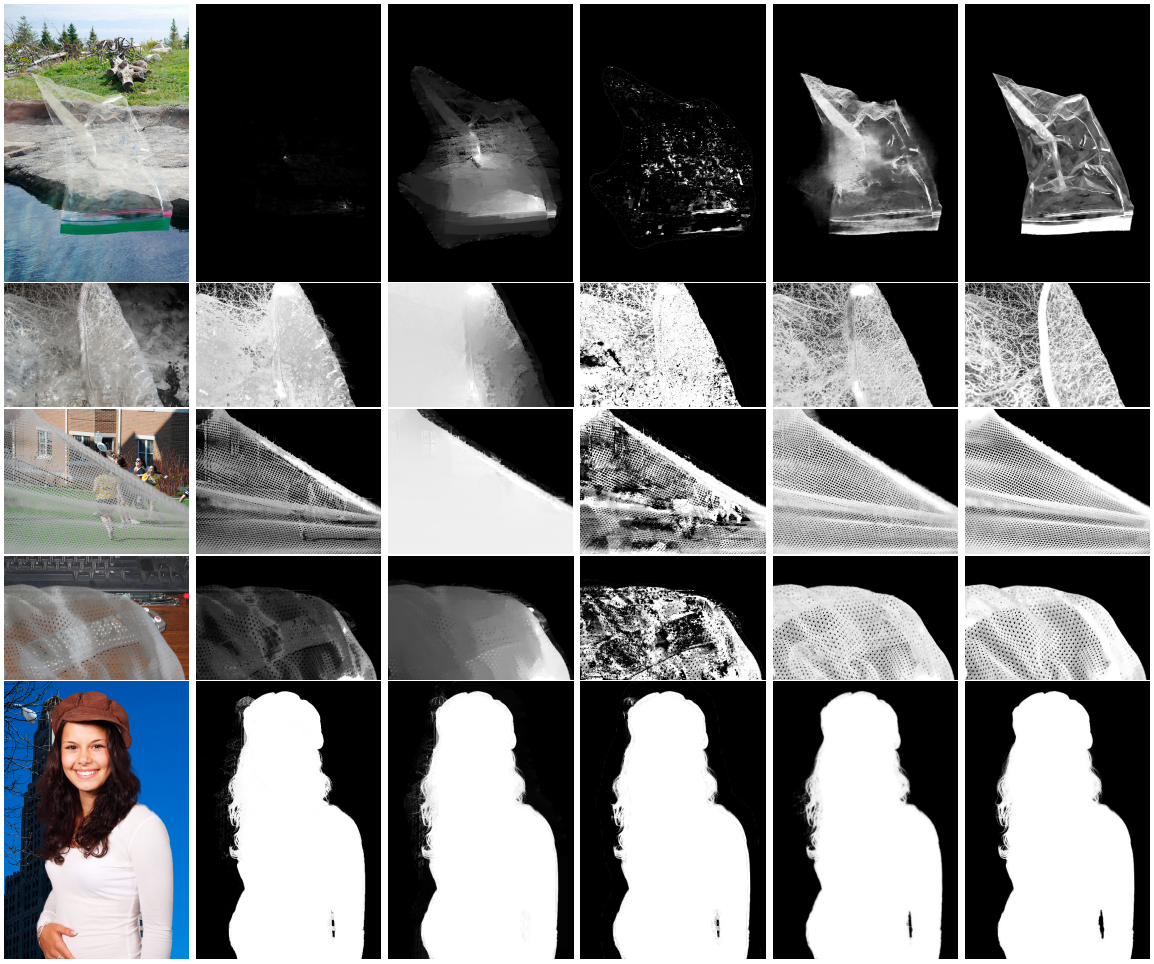
In this section, I will show the image matting results on Adobe dataset and our car-oriented dataset.

4.5.1 Image Matting Results on Adobe Dataset

As other state-of-the-art algorithms were not fine tuned on Adobe dataset with the tri-map images generated by our TGN, it is not fair to use our predict tri-map as input. In the comparison, we took the standard tri-map images provided by Adobe dataset to evaluate the results. We compared MSE and SSIM on the predicted alpha mattes (see table 4.1) and surprisingly found our result came first in MSE comparison and ranked second in the comparison of SSIM. As shown in Fig 4.2, our algorithm has a better matting ability even on images with complex context semi-transparent objects.

Table 4.1: Performance comparison Adobe Matting Dataset.

	KNN	Closed-form	Shared Matting	Ours
MSE	0.01323	0.00974	0.01109	0.00253
SSIM	0.99653	0.99767	0.99747	0.99751



(a) (b) (c) (d) (e) (f)

Figure 4.2: Comparison on Adobe Dataset. (a) RGB Image. (b)KNN results from Chen *et al.* (2013). (c)Closed-form Matting from Levin *et al.* (2007). (d)Shared Matting from Gastal and Oliveira (2010). (e)Our result. (f)Ground truth

.

4.5.2 Deep Image Matting Results on Car-Oriented Dataset

(a) (b) (c)

Figure 4.3: Results from our End-to-End Deep Image Matting on Car-oriented Dataset. (a) RGB Image. (b)Predicted Alpha Matte Image. (c)Compositional Image.

.

Fig 4.3 listed four result sets from car-oriented dataset. Different from alphamatting.com and Adobe dataset, our car-oriented dataset only uses real world images instead of synthetic images. Our dataset dramatically increases the difficulty of image matting since there occurs color overlapping between foreground and background car. Our network can

provide clear and smooth boundary for foreground vehicles and for the semi-transparent regions and transparency is represented by the alpha values. Another improvement of our network is the automatic tri-map generation function provided by GFN where no human work is necessary. The average MSE between our predict alpha matte and ground truth images is only 0.0029, we treat this as a good end point to current project. However, a better performance might be achieved if we applied post-process strategies to the refined results and that will be discussed in Chapter 6.

Chapter 5

Deep Learning Web Structure

In this chapter, I will propose a Java web platform for embedding different deep learning applications. As a web platform, stability and throughput are always the first consideration. In this chapter, I will explain the structure and some leading edge techniques that I applied to optimize this platform.

5.1 Platform Structure

Different from traditional MVC (Model-View-Controller), our proposed deep learning platform (Fig. 5.1) used micro service as the programming structure, which decouples the functionality in a huge web applications to small atomic services (functions in one service cannot be split), so the platform is easier in maintenance and development. Developers don't have to understand the whole platform and only need to pay attention to their own module and the failure of one micro-service won't cause the failure of the whole platform. Meanwhile, the internal mechanism of micro-service structure determines that this platform is possible to be deployed not only on a single server but also on a distributed system

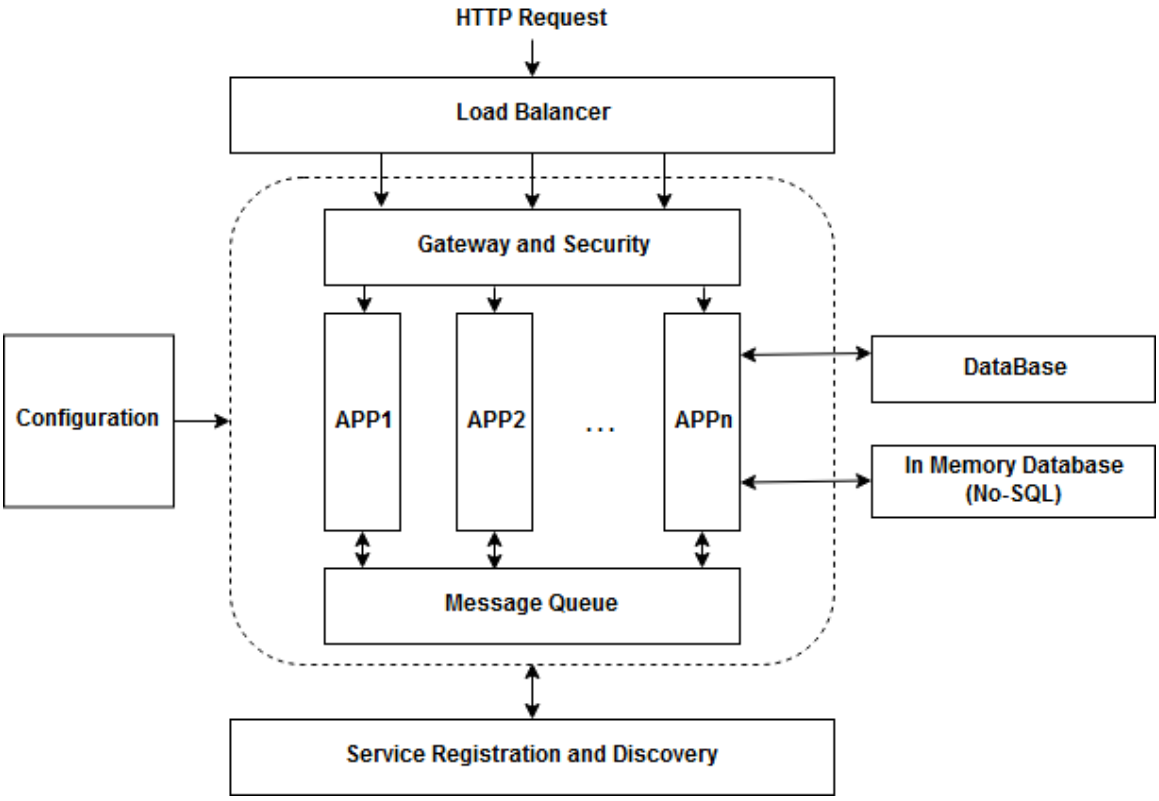


Figure 5.1: Deep Learning Platform Structure.

because service are internal called by request, so the distributor program (gateway) has a chance to forward the request to another cluster by creating a new request.

The key module of micro-service structure platform is service registration and discovery, here we suggest Eureka Server from Spring as the module to register and discover services. Once the back-end program receives a request, smart gateway module will validate the request and forward it to the optimal micro-service according to the information saved in the registration center, which will filter invalid requests like wrong format requests or attack purpose requests, and handle the flood request scenario. This platform is also extendable since all deep learning applications can be registered as services in the registration center.

Load balancer is another important module in our platform. When a deep learning service has multiple copies on multiple physical servers, a load balancer is required to distribute this request by given load balancing strategy to the optimal micro-service. There are several load balancing algorithms, and customers need to customize the load balancing strategy according to usage scenarios. Round Robin algorithm and Weighted Round Robin are two famous load balancing strategies and I will introduce in the following paragraph:

- Round Robin: Application will forward the request to server in order.
- Weighted Round Robin: Since the calculation power of each server is different, customers assign a weight to each hardware (could be server or GPU). When the load balancer receives a request, it will distribute the request according to the assigned weight, which means the servers with more computations power will have a high rate to be used.

5.2 Flow Path of One Request

Here we proposed an asynchronous mechanism of processing a deep learning request (see Fig. 5.2). Once the load balancer service received a request, it will forward that request to a server for processing. During the process, a micro-service need to load the model and process an image but both steps are time consuming. In the asynchronous process, requests are sent to a message queue encapsulated as a message. Once application services are free, they will take a message (or task) from the message queue and process the deep learning task on a free GPU. In order to trace the completion status of a request, an flag representing the progress will be saved in the No-SQL and back-end services will response to user immediately. Front-end applications will periodically send requests to check the

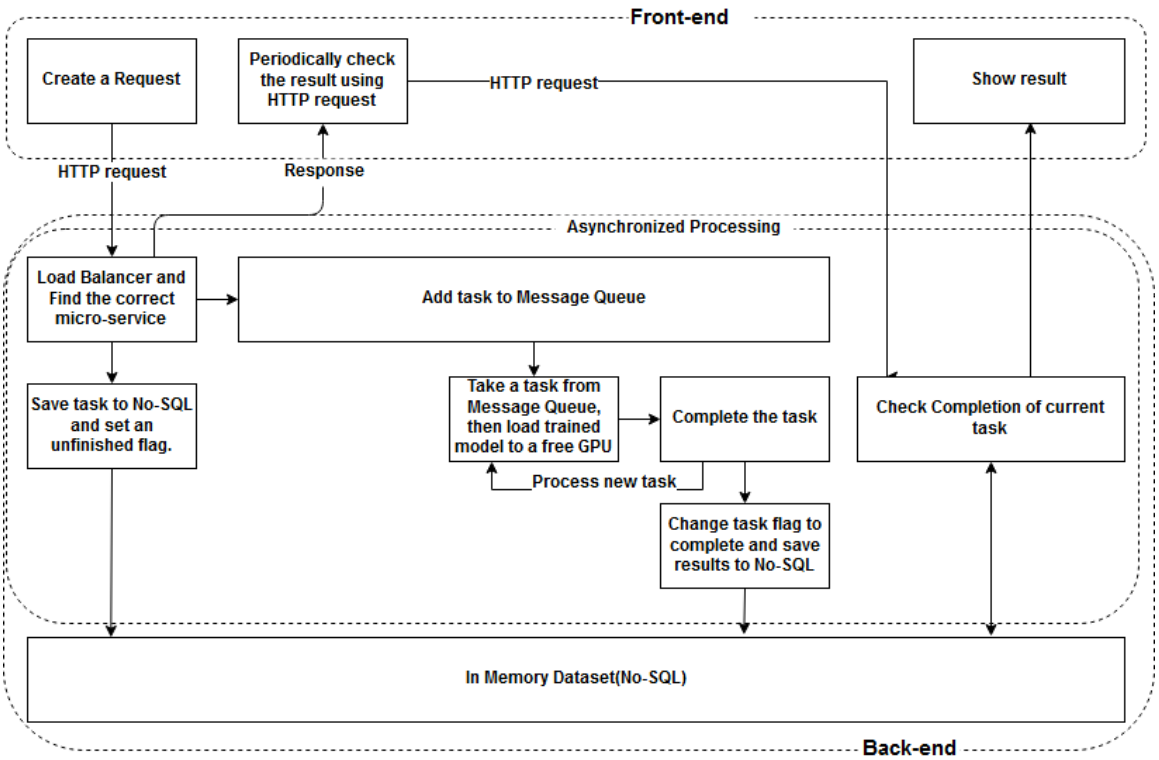


Figure 5.2: Flow Chart for Single Deep Learning Request.

completion of the task until getting the results (see Fig 5.3). Comparing with time spent on processing one image, loading pre-trained model is more time consuming so if deep learning application number is smaller than the GPU number, the most efficient way is to assign a fixed deep learning application to each GPU so deep learning services can avoid the time of loading models.

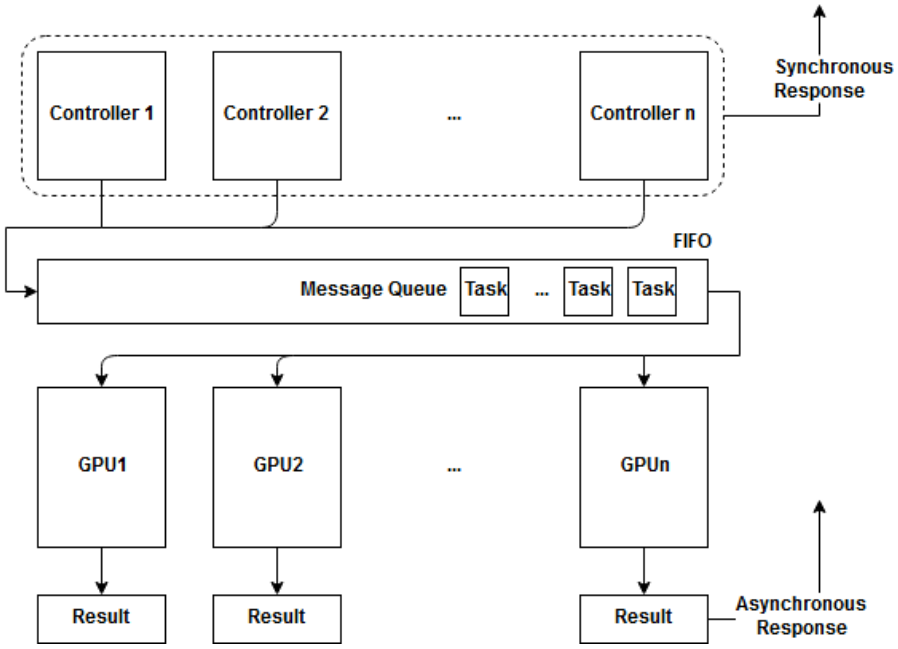


Figure 5.3: Mechanism of Asynchronous response.

Compared with normal web platform, our proposed structure has two advantages. The first one is the asynchronous mechanism because users don't have to pend when they are waiting for the results and a progress component displayed on the screen indicates the progress of current request. Another advantage is the micro-service structure. Deploying new deep learning algorithms is easier since all applications can be registered as an isolated service in this platform. In conclusion, this platform is both user and development friendly and has a higher availability.

Chapter 6

Conclusion and Discussion

In this thesis paper, I proposed an End-to-End Deep Image Matting Network (EDIMN) to solve the image matting problem and a web platform for integrating deep learning applications.

My proposed end-to-end deep image matting network has two stages, the first stage is used to create tri-map automatically by using the idea of instance segmentation. The second stage takes RGB image and the tri-map predicted from the first stage as input and processed by encoder-decoder net and refine net, it creates a precise alpha matte image. Different from previous methods, EDIMN does not require human work to create tri-map and has good performance on not only on synthetic images but also real world images. In order to solve the weakness of current image matting dataset, this thesis paper also provide a real world car-oriented dataset for training purpose which contains 28045 image sets. In our dataset, no foreground objects are repeated so learning based methods could learn more real world features from the dataset than previous datasets. We also proposed a distributive network architecture for embedding different deep learning applications by which integrating new deep learning algorithms becomes easier and its stability when facing high volume requests

is guaranteed due to its asynchronous mechanism.

Moreover, some future extension of our image matting framework could be applied. The first one is to redesign the network to fuse the two stages in EDIMN together as the backbone network in TGN and the encoder stage in image matting network extract high level features from the image so this process is redundant if we apply them in both stages.

The second improvement could be on redesigning the refine net in image matting framwork. After visual comparing the predicted alpha matte images with their ground truth, we find most of the errors occur at the junction of foreground and background. Our network sometimes accidentally recognize foreground pixels as background or vice verse. This problem may be fixed with a deeper refine net borrowing some ideas from semantic segmentation.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, **39**(12), 2481–2495.

Berman, A., Dadourian, A., and Vlahos, P. (2000). Method for removing from an image the background surrounding a selected object. US Patent 6,134,346.

Chen, Q., Li, D., and Tang, C.-K. (2013). Knn matting. *IEEE transactions on pattern analysis and machine intelligence*, **35**(9), 2175–2188.

Chuang, Y.-Y., Curless, B., Salesin, D. H., and Szeliski, R. (2001). A bayesian approach to digital matting. In *CVPR (2)*, pages 264–271.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *International journal of computer vision*, **59**(2), 167–181.

Gastal, E. S. and Oliveira, M. M. (2010). Shared sampling for real-time alpha matting. In *Computer Graphics Forum*, volume 29, pages 575–584. Wiley Online Library.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

Grady, L., Schiwietz, T., Aharon, S., and Westermann, R. (2005). Random walks for interactive alpha-matting. In *Proceedings of VIIP*, volume 2005, pages 423–429.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.

Kohler, R. (1981). A segmentation system based on thresholding. *Computer Graphics and Image Processing*, **15**(4), 319–338.

Levin, A., Lischinski, D., and Weiss, Y. (2007). A closed-form solution to natural image matting. *IEEE transactions on pattern analysis and machine intelligence*, **30**(2), 228–242.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P., and Rott, P. (2009). A perceptually motivated online benchmark for image matting. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1826–1833. IEEE.

Ruzon, M. A. and Tomasi, C. (2000). Alpha estimation in natural images. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 1, pages 18–25. IEEE.

Sathya, B. and Manavalan, R. (2011). Image segmentation by clustering methods: performance analysis. *International Journal of Computer Applications*, **29**(11), 27–32.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *Departmental Papers (CIS)*, page 107.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sun, J., Jia, J., Tang, C.-K., and Shum, H.-Y. (2004). Poisson matting. In *ACM Transactions on Graphics (ToG)*, volume 23, pages 315–321. ACM.

Xu, N., Price, B., Cohen, S., and Huang, T. (2017). Deep image matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2970–2979.