Alpha Matting via Residual Convolutional Grid Network

by

Huizhen Zhang

Thesis submitted to the University of Ottawa
in Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Alpha matting is an important topic in areas of computer vision. It has various applications, such as virtual reality, digital image and video editing, and image synthesis. The conventional approaches for alpha matting perform unsatisfactorily when they encounter complicated background and foreground. It is also difficult for them to extract alpha matte accurately when the foreground objects are transparent, semi-transparent, perforated or hairy. Fortunately, the rapid development of deep learning techniques brings new possibilities for solving alpha matting problems.

In this thesis, we propose a residual convolutional grid network for alpha matting, which is based on the convolutional neural networks (CNNs) and can learn the alpha matte directly from the original image and its trimap. Our grid network consists of horizontal residual convolutional computation blocks and vertical upsampling/downsampling convolutional computation blocks. By choosing different paths to pass information by itself, our network can not only retain the rich details of the image but also extract high-level abstract semantic information of the image. The experimental results demonstrate that our method can solve the matting problems that plague conventional matting methods for decades and outperform all the other state-of-the-art matting methods in quality and visual evaluation. The only matting method performs a little better than ours is the current best matting method. However, that matting method requires three times amount of trainable parameters compared with ours. Hence, our matting method is the best considering the computation complexity, memory usage, and matting performance.

Acknowledgements

I would like to express my gratitude to all those who helped me throughout my entire research.

My deepest gratitude goes first and foremost to my supervisor, Professor Jiying Zhao. In the stages of research, I faced some difficulties. My supervisor always encourages me, supports me, guides me and provides advice to me. Thanks for his constant encouragement and guidance. He is not only the supervisor of my thesis, but also the supervisor of my life. What benefits me is not only his profound knowledge, but also his noble personality.

Second, I would like to express my gratitude to my lab colleagues: Yang Zhou and Lei Chen, for their generous help and support during my study here.

My sincere thanks also goes to my mother, who supports my whole life by herself, who believes me, encourages me, loves me no matter what's happened. Without her, I will never have studied in such a wonderful place.

Last, my thanks would go to my best friend Chen Wei, who gives me a constant source of support and encouragement during my research.

Dedication

This thesis is dedicated to my family.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AdaGrad	Adaptive gradient algorithm
Adam	Adaptive moment estimation algorithm
AI	Artificial intelligence
ASPP	Atrous spatial pyramid pooling
BG	Background
CFM	Closed-Form matting method
CNNs	Convolutional neural networks
Conv	Convolutional
CSM	Comprehensive sampling matting method
DCNN	Deep convolutional neural network
DCNNM	Deep convolutional neural network matting method
DIM	Deep image matting method
ELU	Exponential linear unit
FG	Foreground
GAN	Generative adversarial network
GD	Gradient descent
Gridnet	Grid network
GSM	Global sampling matting method
GT	Ground truth
IFM	Information flow matting method
KNN	K-nearest neighbors algorithm

KNNM	K-nearest neighbors matting method
LBM	Learning based matting method
MAE	Mean absolute error
MRI	Magnetic resonance imaging
MS COCO	Microsoft common objects in context
MSE	Mean absolute error
Pascal VOC	Pascal visual object classes
PReLU	Parametrized rectified linear unit
ReLU	Rectified linear unit
ResNet	Residual network
RM	Robust matting method
RMSProp	Root mean square propagation algorithm
SAD	Sum of absolute differences
SGD	Stochastic gradient decent
SSE	Sum of squared errors
SSM	Shared sampling matting method
TLM	Three-layer matting method

Chapter 1

Introduction

1.1 Alpha Matting

Alpha matting, which is also called natural image matting, aims at accurately extracting the foreground objects and estimating the opacity information of the foreground objects from a complicated natural image.

Alpha matting is an important topic in areas of computer vision. It has various applications, such as virtual reality, digital image and video editing, image and video segmentation, image synthesis, etc. With the recent advances of digital cameras and the flourishing growth of films with virtual scene, using matting techniques to create novel composites or accomplish other editing tasks has attracted more and more attentions from both research communities as well as consumers in recent years.

Mathematically, alpha matting can be expressed using Equ. (1.1):

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i, \tag{1.1}$$

where, I_i is the color of pixel i in a natural image. F_i is the foreground color of pixel i, B_i is the background color of pixel i. The color I_i can be considered as a linear combination of

foreground color F_i and background color B_i. α_i represents the opacity of foreground color of pixel i. It is a value in the range of [0,1], where $\alpha = 0$ indicates a definite background pixel and $\alpha = 1$ denotes a definite foreground pixel. Otherwise it is a mixed pixel. There are some reasons why mixed pixels exist in natural images, i.e. some pixels have an opacity value of $0 < \alpha < 1$: Transparency, sub-pixel structure, optical blur and motion blur.

The purpose of alpha matting is to estimate the *alpha matte*, which is a whole opacity image with corresponding α value for each pixel.

Alpha matting is an inherently under-constrained problem with no exact solution, because it involves seven unknowns: RGB three color components of each foreground color F_i, RGB three color components of each background color B_i, and the α value of pixel i. Meanwhile, it includes three knowns: RGB three color components of I_i for every pixel. However, it contains only three equations. The short of equations and constrains makes alpha matting a challenging problem in image processing since it was proposed.

1.2 User Interaction

To mitigate the difficulty in solving the inherently ill-posed problem, some additional constraints are provided manually by users, such as trimap and scribbles. The trimap roughly segments the natural image into three parts: definite foreground, definite background and unknown regions that consist of a mixture of F and B colors. In the definite foreground region, the alpha values are 1. In the definite background region, the alpha values are 0. Alpha values are computed only for pixels inside the unknown region. Figure 1.1 shows the influence of trimap in alpha matting.

As shown in Figure 1.1, for some matting algorithms, the coarse level of trimaps can influence the accuracy of alpha mattes, the smoother the trimap, the more accurate the alpha matte. However, constructing and calibrating a trimap requires a considerable degree

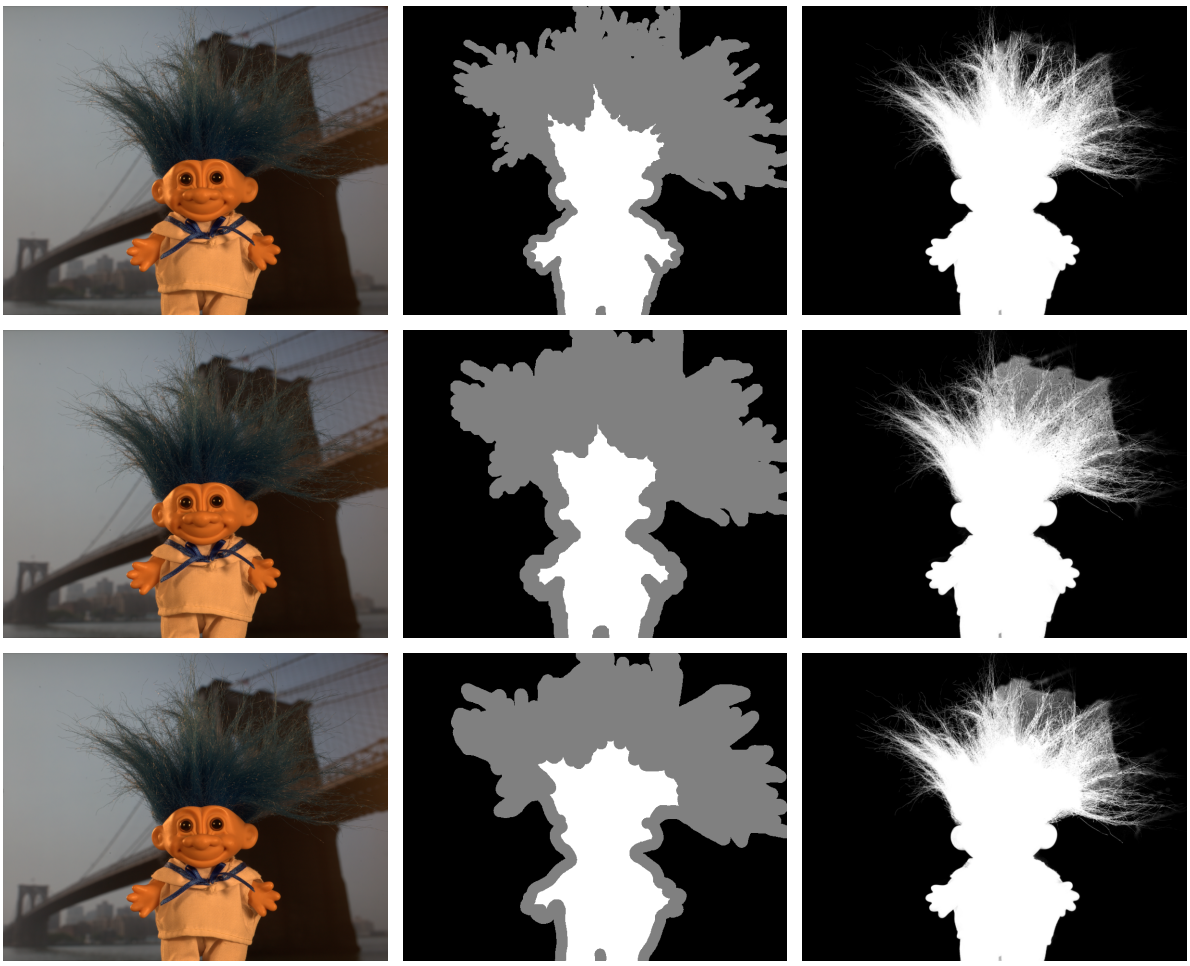


Image Trimap Alpah Matte

Figure 1.1: Example images from the online benchmark [47] for alpha matting. The first column is original image. The second column shows trimaps of different smoothness levels: the first row is a smooth trimap, the second row is a coarse trimap, and the third row is a trimap drawn by user. The third column shows the corresponding alpha mattes generated according to the different trimaps. In the trimaps, the foreground is white, the background is black, and the unknown area is gray. The alpha mattes are obtained by the Three layer matting algorithm [35].

of user interaction, which costs time and energy, especially when the foreground objects are partially transparent, hairy or there is no known foreground object in image, such as the cobweb, the dandelion, the bulb and glass images in Figure 1.2.

Meanwhile, it is almost impossible to manually create an optimal trimap. Hence, some specialists try to obtain the alpha mattes from original images without user-drawn trimaps. Some of them suggest to generate trimaps automatically [2] [20] and others use the scribbles instead of trimaps [34] [60].

Unlike trimap, which carefully defines all pixels in the whole image as foreground pixels, background pixels and mixed pixels, the scribbles only use a few sparse strokes with different colors to represent foreground and background respectively. Figure 1.3 (a) shows an image with user specified foreground and background strokes, where the red strokes represent the foreground objects, and the blue strokes indicate the background. The alpha matte based on scribbles constrains is shown in Figure 1.3 (b), which is extracted by the Iterative BP matting algorithm [60].

User interactions are really necessary for alpha matting. However, since they just help to narrow the unknown regions and add some constrains for calculating, the information provided by user in these way is limited and, as we describe above, expensive. Accurately extracting alpha value from a natural image is still full of challenges.

1.3 Conventional Matting Algorithms

The approaches for solving alpha matting problems have progressed for thirty years. Since new approaches based on deep learning showed up in recent years, we can roughly classify all the alpha matting approaches into two categories: conventional approaches and deep learning based approaches.

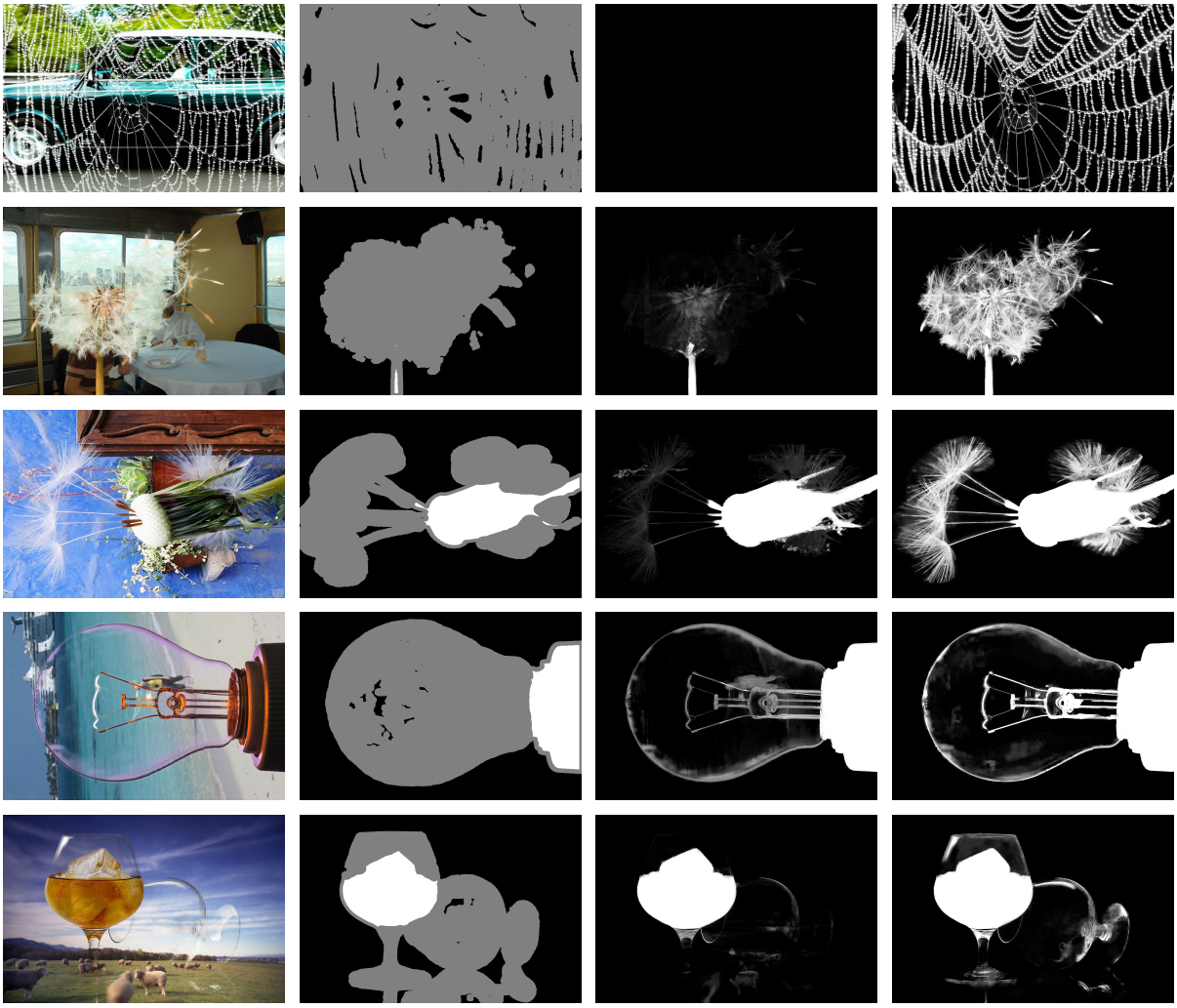
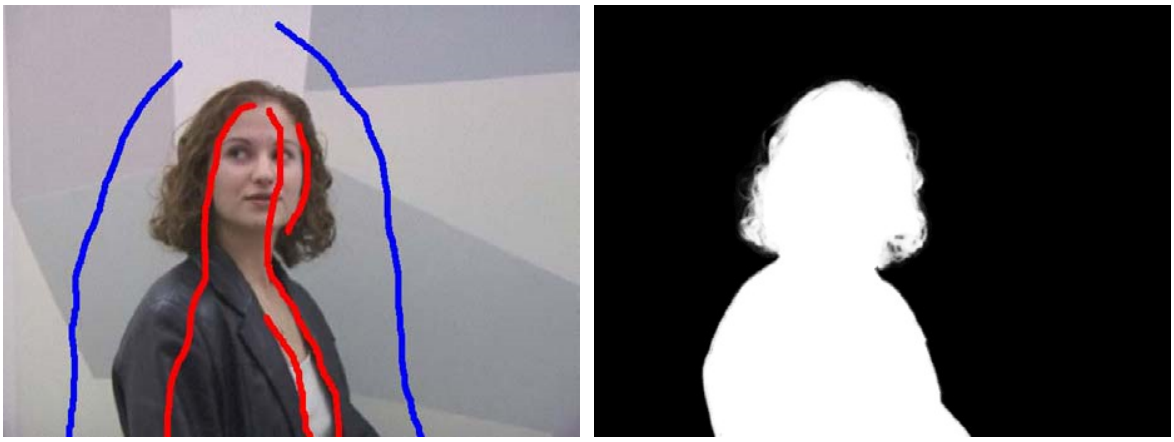


Figure 1.2: The comparison between our method and the Three layer matting [35]. The images and trimaps are from the deep image matting Composition-1k testing dataset [65]. The alpha mattes of the Three layer matting are obtained by running the source code provided by the author.



(a) (b)

Figure 1.3: (a)Original image with user scribbles. (b)Extracted matte by the Iterative BP matting algorithm [60]. The image and alpha matte are from [60].

The conventional approaches for alpha matting can be classified into two categories: sampling-based methods [8], [14], [21], [46], [49], [52], [53], [61], and propagation-based methods [6], [19], [33], [34], [57].

Sampling-based methods collect foreground and background sample pairs from known regions, choose the suitable pairs to represent the true foreground and background colors of the given unknown pixel, and calculate the α value of that pixel based on the estimated colors.

Propagation-based methods propagate α values from known regions to unknown ones relying on the affinity between neighboring pixels.

These conventional approaches mainly rely on color information, spatial position information of pixels, affinity between neighboring pixels, as well as some uncertain assumptions. They are unable to comprehend the semantic information of the natural images. Hence, when the foreground and background color distributions overlap; the background texture is complex; the foreground objects are transparent (e.g., bulb, glass), semi-transparent (e.g., plastic bag, dense mesh, fabrics, veil), perforated (e.g., net, grid, strainer) or hairy

(e.g., hair, feather, fur); or the underlying assumptions are violated, the matting results will deteriorate. We will give an example in the following.

See Figure 1.2. The Three layer matting approach, which was proposed by Li [35], is one of the top conventional approaches. As we can see in the first row of the Figure 1.2, when the image has no known foreground pixels, the conventional matting approach may fail to calculate alpha matte, since most of conventional matting approaches must leverage known foreground and background pixels to calculate alpha matte. Similarly, when the foreground objects are hairy like the ones in row two and row three, the conventional approach may also fail to extract the alpha matte from those hairy regions. The reason is also obvious: in the hairy region, the foreground color information is sparse and dispersive, it is difficult to judge a pixel to be a foreground pixel, a background pixel or a mixed pixel, and it is also difficult to sample a suitable pair to represent the true foreground and background colors of the target pixel. The row four and row five show the situation where a conventional approach encounters images with transparent foreground objects, even with reflection phenomenon in foreground objects (row five). Since most of the conventional matting methods use color information as distinguishing features, when the foreground objects are transparent, they define definite background pixels as foreground ones by mistake (row four). They fail in handling light reflection phenomenon, where they define the reflection part of foreground as a background area.

In fact, the problem mentioned above has plagued image matting research community for a long time, since a lot of foreground objects in natural images are hairy (e.g., animals with fur, people with hair, birds with feather), perforated (e.g., web, net, strainer), transparent (e.g., window, glass, bulb), semi-transparent (e.g., bridal veil, dense mesh, fabrics) and so on. The problem cannot be ignored by any alpha matting researchers. The conventional matting approaches just rely on color and spatial position information as the distinguishing feature, and they cannot comprehend semantic information of images. This inevitably makes the conventional matting approaches fail in dealing with those kinds of

matting problems. Fortunately, the advent of deep learning based matting approaches bring new possibilities for matting problems.

1.4 Deep Learning based Matting Algorithms

The past few years have witnessed a rapid development of deep learning based techniques. As a branch of deep learning, deep convolutional neural networks (CNNs) [32] have also shown excellent performance in massive computer version tasks, such as image classification [31], [56], [58], [66], semantic segmentation [16], [41], [54], [67] and object detection [15], [16], [27], [37], [44], [45], [50]. In image classification task, CNNs define the objects in an image. For semantic segmentation, CNNs identify and segment every object in an image. For object detection, CNNs ascertain the location of objects in an image.

When accomplishing those massive computer vision tasks, CNNs have shown the excellent capability of comprehending the content of the images, as well as extracting the features of the images. So, they are considered to be feasible for solving alpha matting problems.

In recent three years, some matting methods based on CNNs have been proposed. The DCNN [7] method takes the alpha matting results of closed-form matting [34] and KNN matting [6], as well as three-channel original natural image as inputs. It is the first matting method that introduces CNNs into alpha matting problem and has achieved an outstanding performance in matting evaluation. However, the DCNN matting method depends largely on the matting results of conventional matting methods: closed-form matting method [34] and KNN matting method [6], which inevitably retains the matting errors existed in previous matting methods. The automatic portrait matting method [55] only uses CNNs to produce trimap of a person in a portrait image and use closed-form [34] for matting through which matting errors are back propagated to the network. The deep image matting [65]

learns alpha matte directly from three-channel RGB natural image and trimap, and obtains the state-of-the-art performance on the benchmark dataset [47]. However, the network architecture of the deep image matting is complicated. Without the refinement stage, it could generate nearly 26 million parameters which leads to a long training time.

1.5 Thesis Contributions

In this thesis, we propose an alpha matting approach based on CNNs, which has the following four contributions in improving the performance and efficiency of the alpha matting task.

The first contribution is that our method solves the conventional problems that trouble the alpha matting area for decades. As we have mentioned above, conventional matting methods have their own defects in dealing with some special kinds of foreground objects, due to their inherent characteristics. Our method overcomes those defects by comprehending the content of images and learning the natural structure that is presented in images. For example, in the first row of Figure 1.2, the foreground object is a cobweb, which possesses its own textural patterns. Our network can learn that patterns and extract alpha matte from the original image accurately. In the same way, hair, fur, net, grid and other foreground items, they all have their own strong structural and textural features, which can also be learnt and extracted by our network. Our network can also recognize the edges, the region of transparent, the region of light reflection (optional blur) by learning its special structure, respectively. The column four of Figure 1.2 shows the good performance of our method in dealing with those images with special foreground objects.

The second contribution is that we present a full convolutional neural network for alpha matting, which directly learns an end-to-end mapping between image, trimap input and alpha matte, with little pre/post-processing beyond the optimization. Unlike the

DCNN matting [7], which relies on matting results of conventional matting methods, and the automatic portrait matting [55], which uses CNNs to create a trimap, and relies on a propagation-based matting method for matting process, our network learns an alpha matte directly from a given image and trimap pair. Without relying on any conventional matting methods, our method avoids the drawback in conventional matting methods.

The third contribution is that we are the first to introduce grid network to alpha matting and have demonstrated the superiority of this kind of network. Although CNNs have shown impressive success for various computer vision tasks, it does not mean we can use the existing structures directly to solve the matting problems. There are two main reasons: firstly, there is a contradiction between rich matte details and abstract high-level semantic information. For example, with a lot of layers of downsampling, network structures used for classification may get the abstract high-level semantic information from images, but the downsampling operation may led to the lose of matte details, which can not be completely and accurately recovered even if we add some upsampling operation in the following structures. Other low-level computer vision methods, such as image super resolution [10], filtering [64], denoising [62] and deblurring [63], use CNNs to do regression. They also do not fit matting problem because no semantic information is considered. The second reason is the contradiction between a complex network architecture and a limited training dataset. Unlike other computer vision tasks which have rich training and testing dataset, alpha matting problem has a poor dataset because of the difficulty in getting a ground truth. At the beginning, there are only 27 training images and 8 testing images are public on the alpha matting evaluation benchmark website [47], which are used for comparing performance of different kinds of matting methods. Thanks to the contribution of deep image matting [65], which published 431 foreground objects for training and 50 foreground objects for testing, the dataset is much bigger than before, which gives the deep learning based matting methods a chance. Hence, when designing the network for alpha matting, the complexity of network architecture must be considered in case of overfitting.

Our network architecture takes both factors in consideration. By allowing information to flow along multiple paths through the grid network, we obtain both the high-level abstract semantic information of images and the rich matte details of images. Our network has adaptivity, because it can choose paths to pass the information by itself. Meanwhile, our network has scalability. With changing the number of rows or columns of the network, or changing the number of filters in every rows, our network has the ability to fit the size of dataset to a large extent. Even if larger and more diverse datasets are available in the future, our network architecture is still feasible. On the contrary, larger datasets can present challenges for existing deep learning alpha matting methods, which have fixed structures.

The fourth contribution is that our network has a higher efficiency than the best deep learning matting method [65], and has the performance comparable to that method. Our network is intentionally designed with simplicity in mind. Our network structure, which demonstrates to be the best for the alpha matting task with current dataset, contains only 8 million parameters, which is much less than the best matting method, deep image matting method [65], which contains 26 million parameters in the network before adding the refinement. Hence, our network has less complexity and training time. Meanwhile, the alpha mattes obtained by our network have the similar accuracy compared with the best matting method, that makes our method the best one when considering both the efficiency and performance of matting. The numerical and visual comparison will be shown in Chapter 5.

1.6 Thesis Structure

We focus on solving alpha matting problems in this thesis and proposing a novel method: alpha matting via residual convolutional grid network. The rest of this thesis is organized as follows:

In Chapter 2, we introduce the fundamental concepts and techniques of convolutional neural networks(CNNs). In Chapter 3, we review the prior methods used for alpha matting. In Chapter 4, we describe the core components of proposed alpha matting method in detail. In Chapter 5, we present the experimental results on testing dataset, and in Chapter 6 we conclude this thesis and discuss the causes of some phenomena.

Chapter 2

Fundamental Concepts and Techniques

In this chapter, we introduce the fundamental concepts of deep learning, neural networks and convolutional neural networks.

2.1 Artificial Intelligence (AI)

Nowadays, AI is a flourishing field with huge numbers of practical applications and affluent research topics. Some pieces of intelligent software are designed to comprehend speech and images, make diagnoses for patients and automate routine work replacing people. In the early days, AI is used to solve problems that need huge calculation but can be described by a list of formal, mathematical rules. Those problems are intellectually difficult for people but easy for computers.

Gradually, it proves that it is difficult for AI to solve the tasks, which are solved by people intuitively but are hard to be described by a list of completely formal rules. For

example, if we are given a portrait, we can easily recognize the human face in it. However, it is difficult for computers to do that because the knowledge we use to recognize a human face is hard to describe by a list of formal rules. Hence, making the computer system have the ability to solve intuitive problem (such as classification and recognition) becomes a big challenge for AI system.

Some AI projects tried to hard-code some common sense in computer languages, but none of them succeeds, which indicates that an AI system needs the ability to "learn" by itself. This kind of capability is called as machine learning.

2.2 Machine Learning

Machine learning is a field of AI that uses statistical techniques to give computer systems the ability to "learn" from data, without being explicitly programmed. In other word, a machine learning algorithm automatically obtains rules from data and uses the learned rules to do prediction on new data. We should notice that a machine learning algorithm can be automatically improved by experience.

In classic machine learning, the "data" are the representations or features, extracted manually by people from raw data to make machine learning algorithms work. For example, a simple machine learning algorithm called logistic regression can determine whether to recommend cesarean delivery [39]. When logistic regression does that determination, it bases on the diagnostic report given by the doctor, rather than the original MRI scan of the patient. Because, in diagnostic report, the doctor would extract the features from an MRI scan by himself. The feature may be the presence or absence of a uterine scar. What the logistic regression does is mapping the features to the decisions. The logistic regression itself can not deal with the original MRI scan directly, because it can not find the relationship between pixels in the MRI scan and final decisions.

Many AI tasks can be solved by extracting the right set of features, and then providing them to a machine learning algorithm. However, it is a consumption of time and effort to manually extract features from raw data, and to decide which features are more useful than others. Hence, we hope that the machine learning not only can learn the mapping between features and output, but also can automatically extract the effective features by themselves. This kind of machine learning can also be called representation learning or feature learning.

2.3 Representation Learning

Representation learning or feature learning is a set of techniques in machine learning that allows a system to extract the representations or features from raw data by itself and to map the features to the outcomes (decisions or predictions). Comparing to classic machine learning, representation learning has made a great progress by using computer system to replace manual feature extraction. Hence, it provides the possibility in accomplishing some image and video processing tasks with AI system.

However, there are some difficulties in extracting features from raw data, such as how to choose the suitable features and discard unimportant ones. To solve those problems, deep learning is proposed.

2.4 Deep Learning

Deep learning solves the difficulties in extracting the representations (features) in representation learning by exacting high-level abstract features from low-level simpler features. In other word, deep learning enables the computer to build complex concepts out of simpler concepts. To realize this, deep learning uses a cascade of multiple layers of nonlinear

processing units. Each successive layer uses the output from the previous layer as input. In this way, deep learning can learn multiple levels of features that correspond to different levels of abstraction, which forms a hierarchy of concepts.

The following example of computer vision field (see Figure 2.1) explains how a deep learning system can comprehend the concepts of an image by constructing the hierarchy.

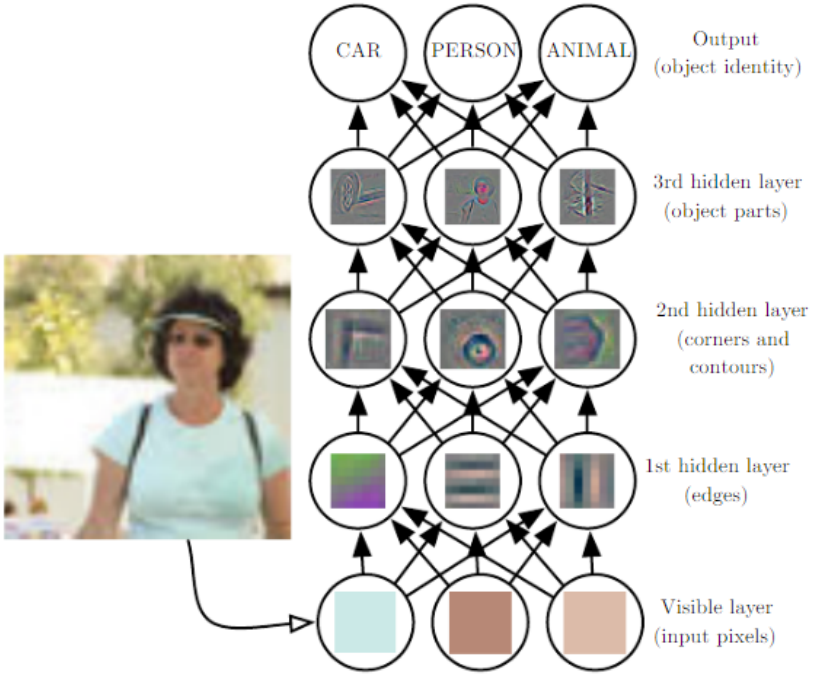


Figure 2.1: A deep learning model for image classification. The image is cited from the book [18].

In fact, in computer vision field, although it is easy for a human being to get details from an image, it is difficult for a computer to do that. This is because an image is encoded as an array of numbers, each of which represents the color value of a pixel. The procedure mapping from a set of pixels to an object identification is very complicated. It seems impossible for an computer system to learn the mapping directly. Deep learning solves this problem by decomposing the desired complicated mapping into a series of simple mappings. Every simple mapping is described by a layer of the model. See Figure 2.1, the input to the

computer system is a set of pixels from an image. We call the input layer as visible layer. Following the visible layer, is a series of hidden layers, which extract increasingly abstract features from the image. The first hidden layer extracts edges from the given pixels. The second hidden layer identifies the corners and extended contours basing on edges. Then the third hidden layer detects object parts basing on the results from the former layer. Finally, these object parts contribute to recognize the objects in the original image.

A number of deep learning architectures, such as deep neural networks, deep belief networks, and recurrent neural networks have been proposed in the fields including computer vision, speech recognition, natural language processing and so on. The prefix "deep" indicates that the depth of the corresponding networks are deep, in order to extract more abstract information.

In this thesis, we focus on solving alpha matting problem, which is a branch of computer vision. Since neural networks is the common representation learning technique applied in computer vision field, we will use deep neural networks to accomplish our alpha matting task. In the following section, we will introduce the related concepts in neural networks.

2.5 Neural Networks

In the following, we will introduce neural networks.

2.5.1 Neuron

The basic computational unit of a neural network is neuron, which has the similar working principle as the biological one in brain. An input x_i is given to the current neuron, multiplied with the corresponding weights w_i. Then the weighted sum of all inputs is calculated, and added with an optional bias b. The result is then passed to an activation function $f(\cdot)$,

which performs a non-linear transformation to generate the output. Figure 2.2 illustrates this procedure and Equ. (2.1) expresses it mathematically:

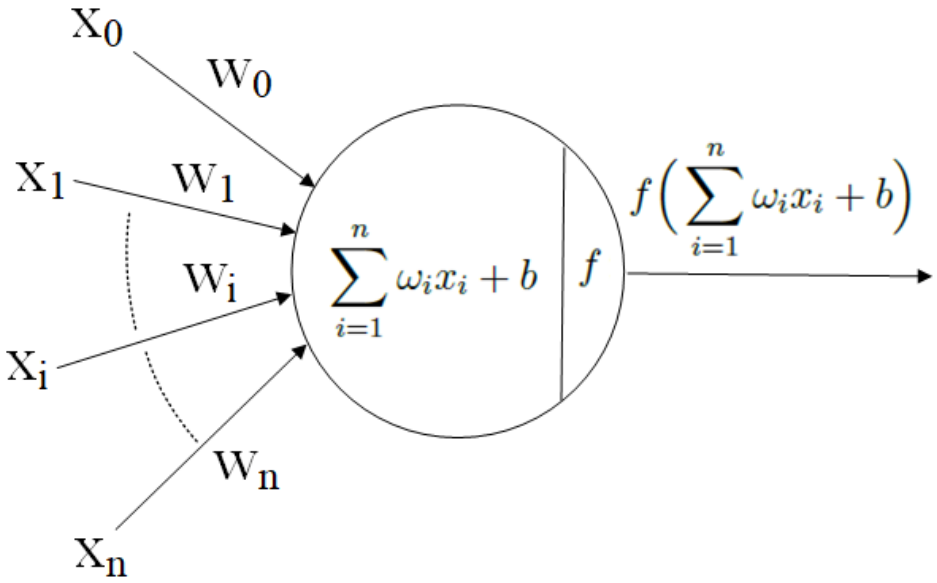


Figure 2.2: A neuron in a neural network.

$$y = f\Big(\sum_{i=1}^{n} \omega_i x_i + b\Big). \tag{2.1}$$

If we represent the weight as a matrix and the inputs and outputs as vectors, the above function can also be expressed as:

$$y = f\Big(\mathbf{w}^\top \mathbf{x} + b\Big), \tag{2.2}$$

where $\mathbf{w} = \{w_1, w_2, ..., w_i, ...w_n\}^\top$, $\mathbf{x} = \{x_1, x_2, ..., x_i, ...x_n\}^\top$, b is a single number.

Sometimes, the activation function is a sigmoid function σ, which takes the addition of the weighted sum of inputs and the optional bias b, and squashes it to range between 0 and 1. There are also many other kinds of activation functions. We will go into more details about different activation functions in the following section.

2.5.2 Neural Network Architectures

Neural Networks are collections of neurons. The neurons arranged in neural networks have the following features:

- Those neurons are connected in an acyclic graph. Cycles are not allowed in the neural networks because that will lead to an infinite loop in the forward pass of a network.
- Neurons are arranged in distinct layers. The first layer is input layer, this layer is the input itself. And the last layer is the output layer, which is responsible for generating the predicted values, i.e. the outputs of the whole neural network. The layers between them are called hidden layers. The outputs of neurons in the previous layer are the inputs of the neurons in the following layer. Neurons within the same layer share no connections. The number of layers decides the depth of the network. In general, deeper neural networks may extract more abstract features and accomplish the task better. But deeper networks are difficult to train and may lead to overfitting and vanishing gradients problems.
- There are various layer types for neural networks, the most common one is the fully-connected layer, where neurons between two adjacent layers are fully pairwise connected. Figure 2.3 is a fully-connected neural network with two hidden layers.

It is very important to apply matrix vector operations for neural networks, so we describe the neural network by matrix in the following. Figure 2.3 shows a three-layer neural network with five inputs and five outputs. We use $a_j^{[i]}$ to represent the jth neuron in the ith layer. The input layer is defined to be the 0th layer, which can be represented as $a^{[0]}$. In our example, input $a^{[0]}$ is a $[5 \times 1]$ vector. We should notice that $a^{[0]}$ is a single data with 5 different factors, such as a single image with 5 pixels.

There is a distinct weight for every connection between two neurons, for the ith layer, all the weighted connection between the current layer and the previous layer can be stored

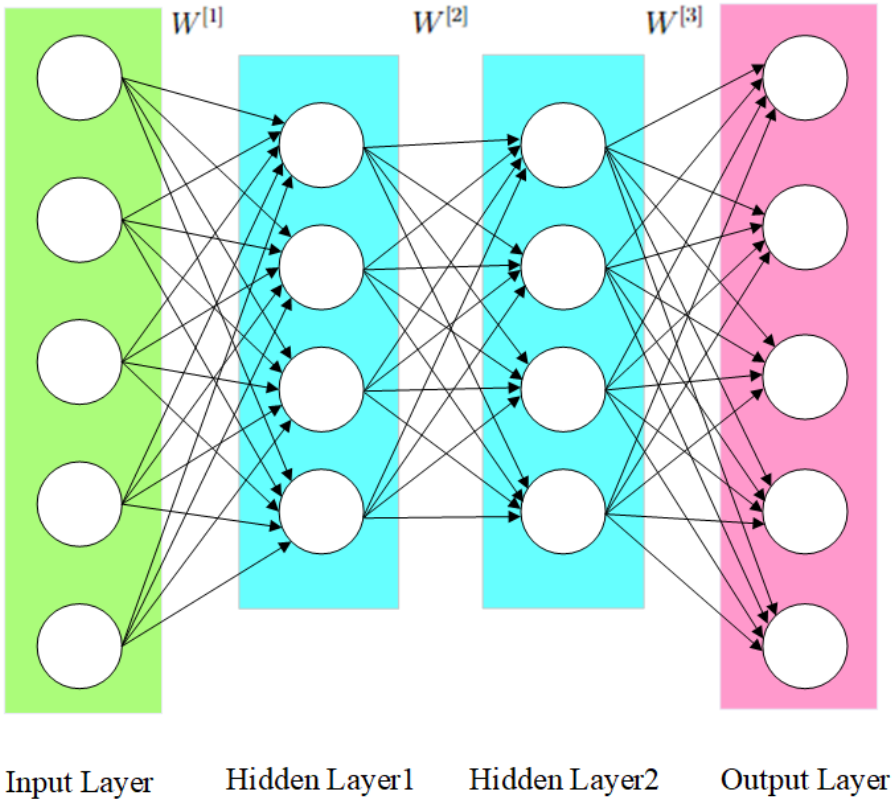


Figure 2.3: A three-layer neural network. When we say N-layer neural network, the input layer is not counted as one layer.

in a single matrix $W^{[i]}$. Every single neuron in the ith layer stores its all weights in a row of $W^{[i]}$. Hence, the size of the weight matrix of every layer in a neural network would be: the number of neurons in the current layer $[i]$ multiplies with the number of neurons in the previous layer $[i-1]$. For example, for the first hidden layer, the weight matrix $W^{[1]}$ would be of size $[4 \times 5]$.

For every neuron, except for the weights, the other parameter is bias b. For every layer, the bias would be in the column vector $b^{[i]}$. The dimension of $b^{[i]}$ would be: the number of neurons in this layer $[i]$ multiplies with 1. For example, the size of vector $b^{[1]}$ would be $[4 \times 1]$.

Using the above neural network structure as example, for a single data, calculating the predicted value through neural network need the following six steps (assuming the activation function is a sigmoid function σ):

$$\begin{aligned} z^{[1]} &= W^{[1]} a^{[0]} + b^{[1]}, \\ a^{[1]} &= \sigma(z^{[1]}), \\ z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]}, \\ a^{[2]} &= \sigma(z^{[2]}), \\ z^{[3]} &= W^{[3]} a^{[2]} + b^{[3]}, \\ a^{[3]} &= \sigma(z^{[3]}). \end{aligned} \tag{2.3}$$

The full forward pass of this three-layer neural network can be seen as three matrix multiplications, interwoven with the application of the activation function. For some specific task, the final neural network layer may not have an activation function.

In the above neural network, $W^{[1]}, W^{[2]}, W^{[3]}, b^{[1]}, b^{[2]}, b^{[3]}$ are the parameters we need to learning from training dataset in training stage. After training, we may fix the values of those parameters and pass forward the testing data to get the predicted value.

If the input is not a single data but an entire batch of training data, we use matrix X to represent the inputs, each single data would be a column of X. Then all the examples would be calculated in parallel.

2.5.3 Activation Function

Firstly, we use the above-mentioned three-layer neural network example to explain the reason why a neural network needs non-linear functions as activation functions for each layer. Given an input vector $\mathbf{x}, a^{[0]} = \mathbf{x}$, we assume the activation functions of the two hidden layers are linear activation functions, e.g., $f(x) = x$.

For the output of the first hidden layer:

$$a^{[1]} = W^{[1]}\mathbf{x} + b^{[1]}. \tag{2.4}$$

For the output of the second hidden layer:

$$\begin{aligned} a^{[2]} &= (W^{[2]}W^{[1]})\mathbf{x} + (W^{[2]}b^{[1]} + b^{[2]}) \\ &= W'\mathbf{x} + b'. \end{aligned} \tag{2.5}$$

In the same way:

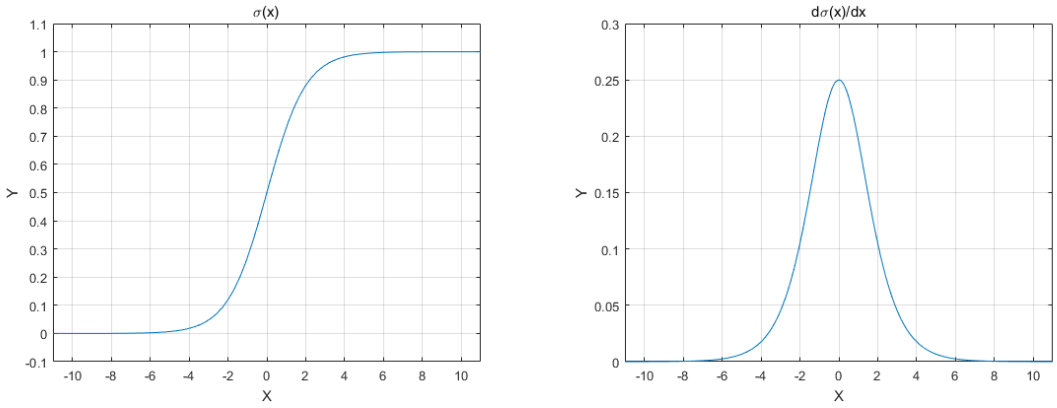
$$\begin{aligned} a^{[3]} &= (W^{[3]}W^{[2]}W^{[1]})\mathbf{x} + (W^{[3]}W^{[2]}b^{[1]} + W^{[3]}b^{[2]} + b^{[3]}) \\ &= W''\mathbf{x} + b''. \end{aligned} \tag{2.6}$$

It turns out that if we use linear functions as the activation functions, or we do not have activation function, then the neural network outputs a linear function of the input. No matter how many layers the neural network has, it just computes a linear function of the input. However, the desired function mapping the input and output is much more complicated than a linear one. Hence, it is very important to introduce non-linear activation functions in neural network, so as to learn complicated function and extract abstract features. We will introduce and compare four common activation functions in the following.

Sigmoid function. The sigmoid function is one of the common used non-linear activation functions. It has the mathematical form:

$$sigmoid : \sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.7}$$

and is illustrated in Figure 2.4(a).



(a) (b)

Figure 2.4: The sigmoid function and its derivative function.

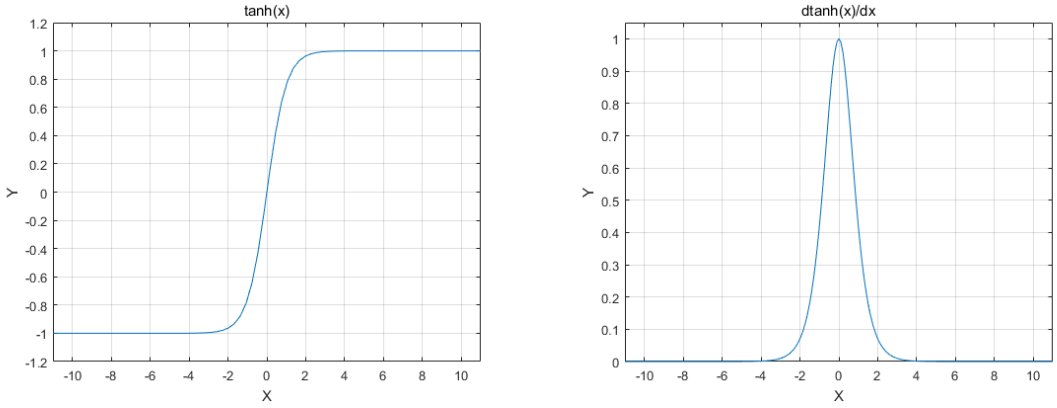
The sigmoid function takes a real-valued number x in the range of $[-\infty, +\infty]$ and squashes it into range $[0, 1]$. It is an S shape function, large positive numbers become 1 and large negative numbers become 0. The sigmoid function is frequently used in the last layer of neural network for the binary classification problem. Because the sigmoid function can classify all real-valued numbers into two categories. However, for the other situations, the sigmoid non-linear function is rarely used. Because of the following two drawbacks:

- Sigmoid function saturates and kill gradients. As we can see in the Figure 2.4(a) and Figure 2.4(b), when the input x tends to infinity, the output $\sigma(x)$ saturates at either 0 or 1, and the derivative at these regions is almost zero. However, when we optimize the neural network, we need to back propagate the derivatives to update the parameters, which we will introduce in the following section in detail. Since the mathematical basis of back propagation is the chain rule of the derivative of calculus,

which means that the derivative of the current layer is the product of the derivatives of the previous layers. These small derivatives may stop the backpropagation, saturate the neurons and finally make the network unable to learn. This phenomenon is also called gradient vanishing.

- Sigmoid outputs are not zero-centered. This causes the data coming into neurons in the later layers of neural network to have non-zero mean, which will affect the gradient. Taking $y = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$ as example, suppose the input is all positive (or negative), then the gradient with respect to w is always positive (or negative), This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights and slow down the speed of convergence.

Hyperbolic tangent function. The activation function that is similar with the sigmoid function, but always works better than the sigmoid function is the hyperbolic tangent function, which is also called the tanh function. The corresponding graph is illustrated in Figure 2.5(a) and the derivative of the tanh function is shown in Figure 2.5(b). Tanh



(a) (b)

Figure 2.5: The tanh function and its derivative function.

function squashes a real-valued input x in the range of $[-\infty, +\infty]$ to the range $[-1, 1]$.

The formula of the tanh function is shown in Equ. (2.8):

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.8}$$

It is mathematically a shift version of sigmoid function, which also can be expressed in this way:

$$\tanh(x) = 2\sigma(2x) - 1. \tag{2.9}$$

Comparing Figure 2.4(b) and Figure 2.5(b), the tanh function has higher gradient value around $x = 0$, which may speed up the gradient descent. Meanwhile, the output of a tanh neuron is zero-centered, overcoming one of the two drawbacks of the sigmoid function. However, the gradient vanishing is still existed because when x tends to infinity the derivative of tanh function also tends to be zero. To overcome the gradient vanishing, one other choice called rectified linear unit, is proposed.

Rectified linear unit. The Rectified linear unit can be called ReLU function for short. The ReLU activation function become very popular in recent years. It is defined as:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.10}$$

which means the ReLU function is a piecewise function. When the input is a positive value in the range of $[0, +\infty]$, the function is a linear function, has the same form as $f(x) = x$. When the input is a negative number in the range of $[-\infty, 0]$, the output is a constant 0. In other words, the activation is simply thresholded at zero. The corresponding graph of ReLU function and its derivative function is shown in Figure 2.6(a) and Figure 2.6(b), respectively. As we can seen in Figure 2.6(b), when $x >= 0$, the derivative is a constant 1, and no saturation any more.

This linear, non-saturating form of ReLU function greatly accelerates the speed of convergence of stochastic gradient descent, and alleviate the vanishing gradient problem,

comparing to the sigmoid and tanh functions [31]. Figure 2.7 shows the superiority of ReLU comparing to the tanh unit. Also, both the sigmoid function and the tanh function involve exponent arithmetic, which are very expensive operations. In contrast, the ReLU is much simpler to be implemented. Those advantages make ReLU a highly recommended activation function. And ReLU is suitable for most neural networks [40].

Unfortunately, ReLU activation function also has its own disadvantages:

- Dead RuLU Problem. It refers to the fact that certain neurons may never be activated, causing the corresponding parameters to never be updated. There are two main reasons that may lead to this situation:

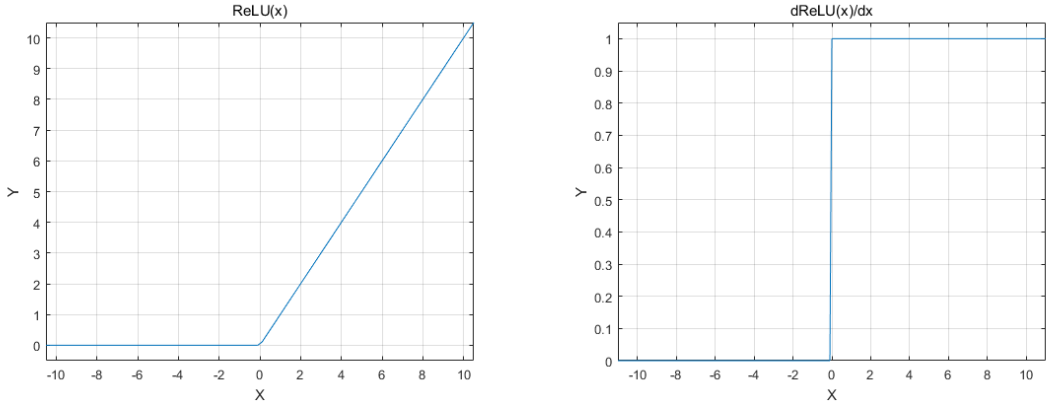
  1. Extremely unfortunate parameter initialization, which is rare.
  2. High learning rate leads to large parameter updating during training, which unfortunately makes the neuron dead.

  The solution is to adopt "Xavier" initialization method, in the meanwhile, to use an algorithm to avoid setting a large learning rate or to adopt "adagrad" [11] to automatically adjust the learning rate.

- The derivative of ReLU activation function is 0 in negative part. And at the point $x = 0$, the ReLU function is non-differentiable. Meanwhile, this function is also non-zero centered.

Despite these two problems, ReLU is still the most recommended activation function when building neural networks.

Leaky ReLU. Leaky ReLU activation function is a variant of ReLU activation function proposed to solve the "Dead ReLU Problem" and reserve the information when $x < 0$. It



(a) (b)

Figure 2.6: The ReLU function and its derivative function.

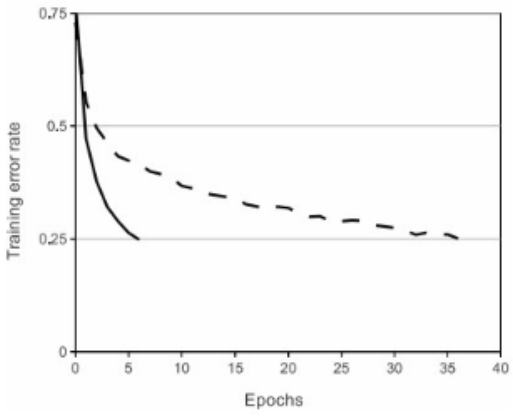
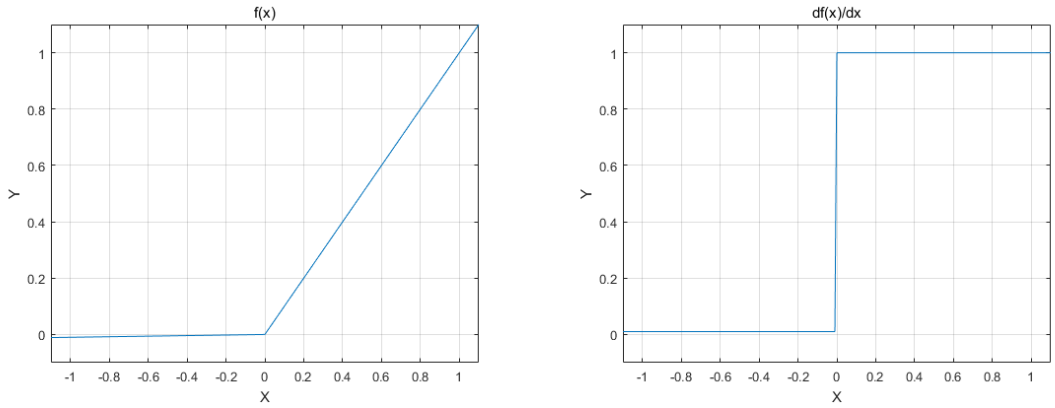


Figure 2.7: A plot from paper [31] indicating the improvement in convergence, comparing with the ReLU and tanh function.

sets the first half of ReLU to be $0.01x$ instead of 0. The formula of the Leaky ReLU is:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01x & \text{otherwise} \end{cases}. \tag{2.11}$$

Its function and derivative function can be found in Figure 2.8(a) and Figure 2.8(b).



(a) (b)

Figure 2.8: The LeakyReLU function and its derivative function.

Another intuitive idea is a parameter-based approach: parametrized ReLU, i.e. PReLU [23], which defines the function as:

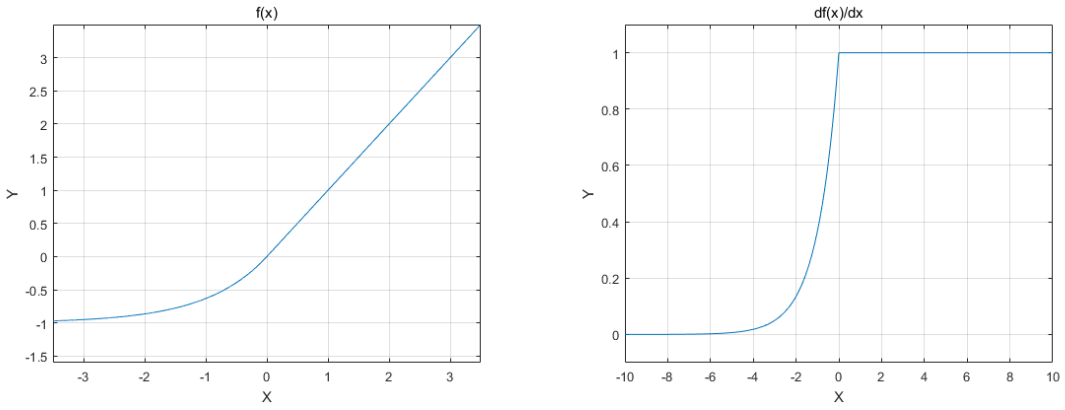
$$\text{PReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise} \end{cases}, \tag{2.12}$$

where α can be learned as a parameter by back propagation during training. In theory, Leaky ReLU and PReLU have all the advantages of ReLU, and overcomes the "Dead ReLU Problem". However, in practice, it is not completely proved that the Leaky ReLU and PReLU are always better than ReLU activation function.

Exponential linear unit. The exponential linear unit (ELU) [9] is also proposed to solve the problem in ReLU activation functions. It is formulated as:

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases}, \tag{2.13}$$

where the coefficient α can be adjusted. The illustration can be found in Figure 2.9(a) and the corresponding derivative graph can be found in Figure 2.9(b). Obviously, the ELU activation function is also a variant of ReLU activation function. Like ReLU, the ELU function alleviates the vanishing gradient problem. Meanwhile, by defining an exponential function for the negative part of x-axis, the ELU function solves the "Dead ReLU Problem" and pushes mean unit activations closer to zero like batch normalization but with lower computational complexity.



(a) (b)

Figure 2.9: The ELU function and its derivative function.

Comparing with the Leaky ReLU, the ELU function has higher computational complexity. Similarly, although the ELU function seems better than the ReLU in theory, there is no obvious evidence that ELU is always superior to the ReLU in practice.

2.6 Training Neural Networks

After introducing the architectures of neural networks, we will focus on how to train the neural networks.

2.6.1 Supervised Learning

In general, machine learning can be classified into two categories: supervised learning and unsupervised learning. Supervised learning learns a function that maps an input to the corresponding output based on labeled data. The labeled data means that some known inputs $\{x^1, x^2, ..., x^i, ..., x^m\}$ are labeled by corresponding known outputs $\{y^1, y^2, ..., y^i, ..., y^m\}$, where x^i is the ith sample in the data. The whole data has m samples. At the beginning, we feed the system with the known input-output (label) pairs, then the error term, the degree to which the system fails to produce the label, is calculated. After that the error term is used as feedback to correct the learning process. When the error is small enough, the system is fixed, which can be used to predict new data. Neural networks belong to this category. In unsupervised learning, the data are unlabeled. In this thesis, we will focus on supervised learning.

2.6.2 Parameters and Hyperparameters

In machine learning, there are two kinds of parameters: model parameter and model hyperparameter. A model parameter is a configuration variable that is internal to the model and whose value can be learnt from data. In neural networks, the collections of all weights w and all bias b are parameters. In fact, the process known as model training is using an optimization algorithm to estimate the optimal parameters according to the historical training data.

A model hyperparameter is a configuration that is external to the model and whose value cannot be learnt from data, but need to be specified by the practitioner manually. The model hyperparameters are often used in process to help estimate model parameters. In neural networks, the learning rate for training the network; and in k-nearest neighbor classification model, the value of k are all model hyperparameters.

We should notice that we can not use the testing dataset to find the suitable hyperparameters, because the testing dataset should never be used until the end. Hence, to find the suitable hyperparameters, we should utilize training dataset effectively. We will introduce the way to utilize training dataset in the following.

2.6.3 Dataset

The dataset is very important for machine learning, since the model determines its parameters and hyperparameters totally based on dataset. In general, the dataset can be split into two parts: training dataset and testing dataset. The training dataset is used to train the model and to learn the optimal parameters of the model. The testing dataset is not used until we have learnt the optimal parameters and finished the training. The testing dataset is used to measure the performance of our trained model.

Sometime, we further split the training dataset into two parts: a slightly smaller training dataset, and a validation dataset. As we mention above, there are hyperparameters in the model, and the values of the hyperparameters cannot be learnt directly from training dataset. The way to find the best values of hyperparameters, is trying out different values on them and using a "fake" testing dataset to estimate the effect. The validation dataset can then be used as the "fake" testing dataset.

2.6.4 Loss Function and Cost Function

In order to train the parameters (all weights w and bias b) for a neural network model, a training set with m training examples is given: $\{(x^1, y^1), ..., (x^i, y^i), ..., (x^m, y^m)\}$. We hope to find the parameters w and b, so that at least on the training set, the prediction we have, which can be expressed as $\hat{y^i}$, is consistent with the ground truth label y^i in the training set. When we achieve this objective, we can say that our model learns the function that maps the input with the output well. Or we can say that our model has done a good prediction based on the training dataset.

To measure how well the prediction is doing by the model, the loss function, which calculates the difference between prediction result $\hat{y^i}$ and the ground truth label y^i, is used. Of course, the prediction result $\hat{y^i}$ depends on inputs x^i, weights w and bias b. A "good" model of a dataset will have small prediction errors, and therefore produce small loss function values. Determining the "best" model is equivalent to find suitable parameters w and b, to minimize the loss function.

In our task, we hope the prediction of alpha matte is consistent with the ground truth alpha matte, at least in training dataset. Hence, we focus on the loss functions which evaluate the differences between two images. One of the simplest possibilities in comparing images is to compare them pixel by pixel and add up all the differences. Hence, we use two matrices, I_{pre} and I_{gt}, to represent the predicted alpha matte and the ground truth alpha matte, respectively. For an image of size $m \times n$, I_{pre} is an $m \times n-$ dimensional matrix, so is the I_{gt}. Each entry in the matrix represents the grey value for a single pixel in the image. In the following we introduce some common choices for loss functions.

SAD, L1-norm and MAE. SAD is the short name of the sum of absolute differences. It measures the similarity between image blocks by taking the absolute difference between each pixel in the ground truth image block and the corresponding pixel in the predicted

image block. These differences are summed to create a metric of block similarity. It can also be seen as the L1-norm loss function. The L1-norm loss function also minimizes the sum of the absolute differences between the target image and the estimated image. The SAD takes the formula the same as L1-norm loss function:

$$\mathrm{SAD}(I_{pre}, I_{gt}) = \mathrm{L1}(I_{pre}, I_{gt}) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\Big|I_{pre}(i,j) - I_{gt}(i,j)\Big|, \tag{2.14}$$

where $|\cdot|$ denotes the absolute value, $I_{gt}(i,j)$ denotes the pixel value of the ith row, jth column in the ground truth image, $I_{pre}(i,j)$ denotes the pixel value of the ith row, jth column in the predicted image.

If we take the mean of the L1-norm loss function, it is another loss function, named MAE, the short for mean absolute error. It is formulated as:

$$\mathrm{MAE}(I_{pre}, I_{gt}) = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\Big|I_{pre}(i,j) - I_{gt}(i,j)\Big|. \tag{2.15}$$

SSE, L2-norm and MSE. SSE is the short name of the sum of squared errors, it measures the difference between original image and the comparing image by calculating the sum of the squares of the error between each pixel in the ground truth and the corresponding pixel in the predicted image. It has the same function as L2-norm loss function. The L2-norm loss function also minimizes the sum of the square of the differences between the target image and the estimated image. The SSE and L2 loss function take the following form:

$$\mathrm{SSE}(I_{pre}, I_{gt}) = \mathrm{L2}(I_{pre}, I_{gt}) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\Big[I_{pre}(i,j) - I_{gt}(i,j)\Big]^2. \tag{2.16}$$

MSE, an abbreviation of the mean squared error measures the average of the squared difference between the target image and the estimated image. It takes the following formula:

$$\mathrm{MSE}(I_{pre}, I_{gt}) = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\Big[I_{pre}(i,j) - I_{gt}(i,j)\Big]^2. \tag{2.17}$$

We use $L^i(I_{pre}, I_{gt})$ to express the loss function of the ith sample in the training set. Then the average of the sum of the all loss functions for the training set is called cost function, which is defined as the following:

$$\mathrm{C}(I_{pre}, I_{gt}) = \frac{1}{m} \sum_{i=1}^{m} L^i(I_{pre}, I_{gt}), \tag{2.18}$$

where m is the number of the samples in the training set. In general, we do not tell the difference between loss function and cost function. Hence, the aim of our training algorithm is to minimize the cost function.

In the following section, we will introduce some efficient optimization algorithms that attempt to find a local or global minima of the cost function.

2.6.5 Optimization

The process that neural network tunes its parameters so as to minimize the cost function is called optimization. The smallest overall value of the cost function is a global minimum. The cost function is a function has a large number of variables and it can be seen as a surface in a very high-dimensional space. Every dimension is a tunable parameter. Figure 2.10 is a sketch map of a cost function in three-dimensional space. In theory, we hope to find the global minima by training the neural network, however, in practice, it is very difficult to really find the global minima, so it is also acceptable to find a small enough value of cost function and use the parameters at that point to predict results of testing dataset.

Gradient Descent (GD). Gradient descent is an efficient optimization algorithm that attempts to find a local or global minima of a function. At the beginning, we generate a random weight matrix W, it can be seen as an imaginary ball on the cost surface in the high-dimensional space. The algorithm drives the ball to roll down the surface according to the

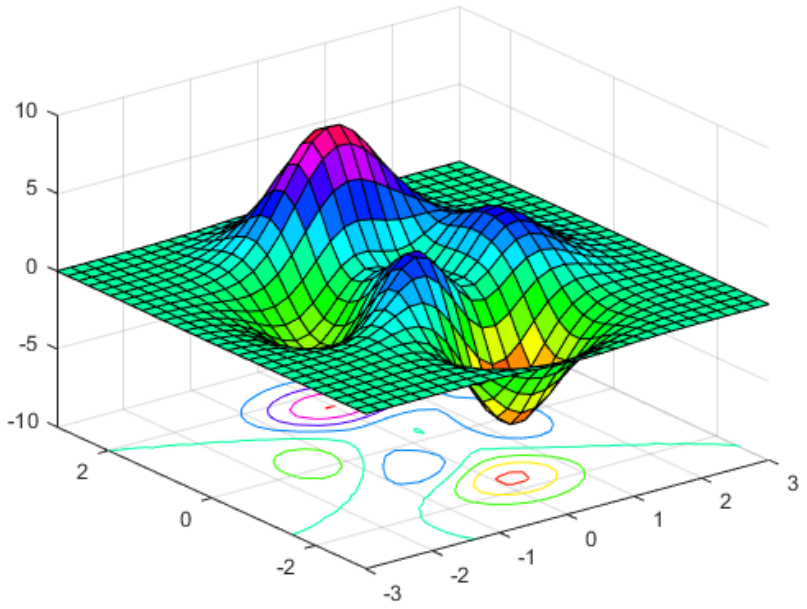


Figure 2.10: The cost function in a 3D space.

gradient at its position. The interactive update process can be expressed mathematically:

$$w_{t+1} = w_t - \eta \nabla C_t(w), \tag{2.19}$$

where w_t denotes the parameters vector in the t-th step. η is the step size of every iteration, which is a small, positive parameter also called as learning rate. $C_t(w)$ is the value of cost function in the t-th iteration. $\nabla C_t(w)$ is a gradient vector. For example, if the cost function has n dimensions variables $w_1, ..., w_j, ..., w_n$, then the gradient is simply the vector of partial derivatives in each dimension:

$$\nabla C_t(w) = (\frac{\partial C_t(w)}{\partial w_1}, ..., \frac{\partial C_t(w)}{\partial w_j}, ..., \frac{\partial C_t(w)}{\partial w_n})^\top. \tag{2.20}$$

Gradient descent algorithm updates every parameter in every step (one iteration), by computing the loss function over the entire training dataset. Hence, GD updates slowly and has a high computation complexity. When further tweaks to the parameters produce little or zero changes in the cost, convergence achieves. The coordinates where the imaginary ball settles is the optimized parameters.

Stochastic Gradient Descent (SGD). At the beginning, SGD [4] is the optimization algorithm that uses only one single training data to compute the gradient at every step. In recently, we prefer to use it to represent mini-batch gradient descent algorithm, which uses the gradient of a batch of sampled training data to approximate the true gradient of the whole training dataset. This raises two concepts: "epoch" and "mini batch size". The "epochs" is the number of times the entire training dataset will be traversal by the training algorithm. The "mini batch size" is the size of the mini batches to use when sampling. In each epoch, we firstly randomly shuffle the training data, and then partition it into mini batches of the preset size. Then we update the network parameters for one iteration, by applying gradient descend algorithm on a mini batch of training dataset.

In order to distinguish SGD and GD easily, we use $J(w)$ instead of $C(w)$ to express the objective function:

$$J(w) = \frac{1}{m} \sum_{i=1}^{m} L^i(w), \tag{2.21}$$

where $L^i(w)$ is the loss function of the i-th sample in the mini training batch. m is the size of the mini training batch. For the t-th iteration, the SGD update procedure is given by:

$$w_{t+1} = w_t - \eta \nabla J_t(w), \tag{2.22}$$

where $\nabla J_t(w)$ is the gradient of the objective function, w is the parameter vector, η is the learning rate.

Momentum. Momentum algorithm [43] utilizes the concept of Momentum in physics. It assumes the imaginary ball has inertia when it rolls down from the hill, which means the direction calculated previously will influence current direction. This prevents the imaginary ball from oscillating and encourage the ball to keep traveling in the same direction (towards the optimum). The momentum update is given by Equ. (2.23):

$$\begin{aligned} v_t &= \eta \nabla J_t(w) + \gamma v_{t-1} \\ w_{t+1} &= w_t - v_t, \end{aligned} \tag{2.23}$$

where v_t is the velocity vector in the t-th iteration, which is of the same dimension as the parameter vector w_t. v_t combines the gradient of the current objective function with the influence of the previous gradient. η is the learning rate and γ is the momentum term, it is usually set to 0.9.

Adagrad. Adagrad [11] is a gradient-based optimization algorithm that adapts the learning rate to the parameters. The learning rate is decreased for parameters associated with frequently occurring features and is increased for parameters associated with infrequent features. Hence, it is well-suited for dealing with sparse data. For every parameter w_j, Adagrad modifies the learning rate η at each step t by:

$$w_{t+1,j} = w_{t,j} - \frac{\eta}{\sqrt{G_{t,jj} + \epsilon}} g_{t,j}, \tag{2.24}$$

where $g_{t,j} = \frac{\partial J_t(w)}{\partial w_j}$ is the partial derivative of the objective function w.r.t. the parameter w_j at step t. ϵ is a smoothing term that avoids the denominator to be zero, which is always on the order of $1e-8$. $G_t \in \mathbb{R}^{n \times n}$, is a diagonal matrix. Every diagonal element jj in G_t is the sum of the squares of the gradients w.r.t.w_j, from the first iteration to the current iteration t. $G_{t,jj}$ can be defined as:

$$G_{t,jj} = \sum_{\tau=1}^{t} g_{\tau,j}^2, \tag{2.25}$$

where g_τ is the gradient at τ iteration.

Adagrad algorithm eliminates the annoyance in manually tuning the learning rate. But the accumulation of the squared gradients in the denominator causes the learning rate to shrink during training and become infinitely small at some point. The following algorithms aim to resolve the continual aggressive decay of learning rates throughout training.

RMSProp. RMSProp [59] is also an adaptive learning rate algorithm proposed to resolve the radically diminishing learning rates in Adagrad algorithm. Instead of accumulating all

the past squared gradients, the RMSProp calculates the moving average of the past squared gradients for each weight. And then divides the gradient by the root of the mean squared gradient. This is why its called RMSprop (root mean square). At the t-th iteration, the moving average is $E[g^2]_t$, which depends on the current gradient and the previous average:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2. \tag{2.26}$$

The update procedure is given by:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t. \tag{2.27}$$

The learning rate is initialized as 0.001, and ϵ defaults as $1e^{-8}$.

Adam. Adaptive Moment Estimation (Adam) [30] algorithm, essentially, it is an RMSprop with momentum term, which dynamically adjusts the learning rate of each parameter by using the first moment (the mean) estimation and the second moment (the uncentered variance) estimation of gradient. The advantage of Adam is that, after bias correction, the learning rate of each iteration has a certain range, so that the parameters are relatively stable. Formula is as follows:

$$m_t = \mu m_{t-1} + (1 - \mu) g_t, \tag{2.28}$$

$$n_t = \nu n_{t-1} + (1 - \nu) g_t^2, \tag{2.29}$$

where m_t and n_t are the first moment estimation and the second moment estimation of gradient respectively, which can be seen as the estimation of expectation $E[g]_t$ and $E[g^2]_t$. Then the bias-corrected first and second moment estimation are computed, which can be approximated as unbiased estimation of the expectation $E[g]_t$ and $E[g^2]_t$.

$$\hat{m_t} = \frac{m_t}{1 - \mu^t}, \tag{2.30}$$

$$\hat{n_t} = \frac{n_t}{1 - \nu^t}, \tag{2.31}$$

where μ has the default value 0.9 and ν has the default value 0.999. Finally, the bias-corrected first moment estimation $\hat{m}_t$ and the bias-corrected second moment estimation $\hat{n}_t$ are used to update the parameters:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{n}_t} + \epsilon}\hat{m}_t, \tag{2.32}$$

where ϵ defaults as $1e^{-8}$.

Since $-\frac{\hat{m}_t}{\sqrt{\hat{n}_t}+\epsilon}$ forms a dynamic constraint on the learning rate and has a clear range, the parameter can be updated more smoothly. In practice, Adam works well and is superior to other adaptive learning rate algorithms, hence, Adam is the default recommended optimizer for most neural networks.

2.6.6 Backpropagation

When a neural network works, it receives an input x^i from the input layer. Then it passes the information through the following hidden layers, and finally produces an output $\hat{y^i}$. This produce is so-called forward propagation. In training procedure, the output $\hat{y^i}$ and the ground truth label y^i will be fed to the loss function $L(\hat{y^i}, y^i)$ and produce an scalar error value. The backpropagation algorithm makes the error value backward flow from the output layer, where the scalar error value produced, to the input layer. Along with the error value passing backward through the network, the gradients of all parameters are recalculated and all of the parameters are updated according to the gradients.

The backpropagation algorithm is a way of computing gradients through recursive application of chain rule of calculus. The chain rule states that the derivative of a composite function $f(g(x))$ can be expressed as the product of derivatives of the related functions. For example, let f and g are two functions in real number domain, $h = g(x)$, $z = f(h) = f(g(x))$. Then according to the chain rule, the derivative of z with respect to x can be expressed using:

$$\frac{dz}{dx} = \frac{dz}{dh} \cdot \frac{dh}{dx} = f'(h)g'(x). \tag{2.33}$$

We will use an example to demonstrate how the back propagation algorithm works based on the chain rule. Assuming we have a neural network like Figure 2.11. As we explained in the previous section, in the forward propagation procedure, the information is firstly passed from the input layer to the hidden layer and then to the output layer. Use neuron o_1 as examples:

$$\begin{aligned} in_{o_1} &= w_5 * h_1 + w_6 * h_2 + b_3 * 1, \\ out_{o_1} &= \frac{1}{1 + e^{-in_{o_1}}}, \end{aligned} \tag{2.34}$$

when the activation function is sigmoid function.

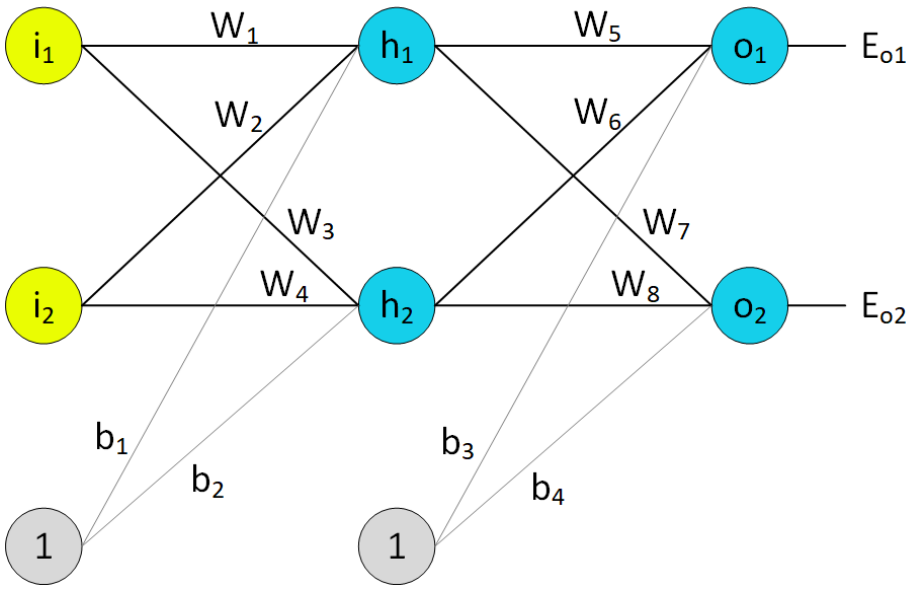


Figure 2.11: An example neural network for backpropagation illustration.

Assume the target value of the out_{o_i} is the $target_{o_i}$, then the total error E_{total} is the sum of E_{o_i}:

$$\begin{aligned} E_{o_1} &= \frac{1}{2}(target_{o_1} - out_{o_1})^2, \\ E_{o_2} &= \frac{1}{2}(target_{o_2} - out_{o_2})^2, \\ E_{total} &= E_{o_1} + E_{o_2}. \end{aligned} \tag{2.35}$$

Then we demonstrate the backpropagation algorithm. Using w_5 as an example, if we try to calculate that how much w_5 contributes to the total error E_{total}, we will calculate

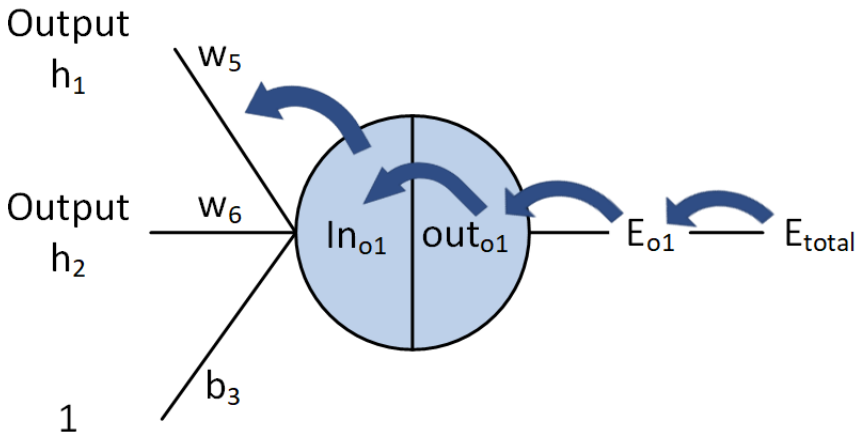


Figure 2.12: An example of calculating the partial derivative of the total error E_{total} with respect to w_5 by using backpropagation algorithm.

the partial derivative of the total error w.r.t. w_5. Based on the chain rule, it is given by:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial E_{o_1}} * \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial in_{o_1}} * \frac{\partial in_{o_1}}{\partial w_5}. \tag{2.36}$$

Calculating every product factor in the above equation, the first product factor is:

$$\frac{\partial E_{total}}{\partial E_{o_1}} = 1; \tag{2.37}$$

the second factor is calculated in this way:

$$\begin{aligned} \frac{\partial E_{o_1}}{\partial out_{o_1}} &= 2 * \frac{1}{2}(target_{o_1} - out_{o_1})^{2-1} * (-1) \\ &= -(target_{o_1} - out_{o_1}); \end{aligned} \tag{2.38}$$

the third factor can be calculated as:

$$\begin{aligned} \frac{\partial out_{o_1}}{\partial in_{o_1}} &= (-1) * (1 + e^{-in_{o_1}})^{-2} * (e^{-in_{o_1}}) * (-1) \\ &= out_{o_1}(1 - out_{o_1}); \end{aligned} \tag{2.39}$$

and the last product factor is:

$$\begin{aligned} \frac{\partial in_{o_1}}{\partial w_5} &= \frac{\partial (w_5 * h_1 + w_6 * h_2 + b_3 * 1)}{\partial w_5} \\ &= out_{h_1}. \end{aligned} \tag{2.40}$$

We multiply these five factors, get the value $\frac{\partial E_{total}}{\partial w_5}$, and update the parameter w_5 using gradient descend algorithm:

$$w_5' = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}. \tag{2.41}$$

In the same way, we can update the other parameters in the neural network, the Figure 2.12 illustrates this procedure.

In fact, the backpropagation algorithm refers to the way of calculating the gradients of every parameter. The SGD-family algorithms described in Section 2.6.5 are actually the optimization algorithms that guide the parameters update and optimize the neural network according to the gradients.

2.7 Convolutional Neural Networks

Convolutional neural networks (CNNs) have shown excellent performance in solving computer vision problems in recent years. CNNs have similar working principle with the ordinary neural networks. The reason why we invent CNNs is that regular neural networks do not scale well to full images. For example, when the input image of a fully connected neural network is of size $320 \times 320 \times 3$, a single neuron in the first hidden layer would have $320 \times 320 \times 3 = 307200$ weights. If we have 10 neurons in the first hidden layer, it would be $3,072,000$ weights just for one layer. It is clear that the parameters would add up quickly as the neural network going deeper. To control the number of parameters in a reasonable range and extract the features of high resolution images, CNNs are proposed. We will introduce the concept of CNNs in the following.

2.7.1 Convolutional Neural Networks Architectures

Since the inputs of Convolutional Neural Networks are images, the CNNs constrain the architecture in a way different with regular fully connected neural networks: the layers of

a CNN have neurons arranged in three dimensions: width, height, depth; A neuron in a layer only connects to a small local region of the previous layer, instead of all the neurons. Figure 2.13 is a visualization.

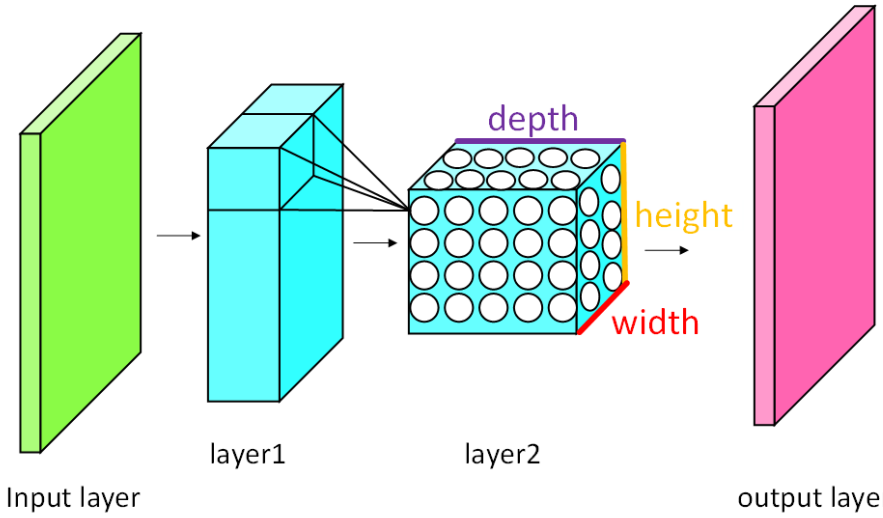


Figure 2.13: An Illustration of CNN. Every layer in the CNN transforms an input 3D volume to an output 3D volume.

The layers building CNNs architectures have two main types: convolutional layer, and pooling layer. These layers are stacked to form a Convolutional Neural Network architecture. Among them, convolutional layer contains parameters and pooling layer does not. In the following section, we will introduce the two types of layers, respectively.

2.7.2 Convolutional Layer

The convolutional layer is the most important building block of a CNN, which undertakes most of the computation. As we introduced in the previous section, the neurons in the convolutional layer are arranged in 3−dimensions as a cuboid. We slice the cuboid in depth, and get the 2−dimensional (width and height) slice of depth (depth slice). Every neuron in one depth slice connects to an equivalent size local region in the input volume, assuming

it is a $11 \times 11 \times 3$ region. To control the number of parameters in CNNs, each neuron in the same depth slice shares the weights and bias with others. For example, the convolutional layer is of size $32 \times 32 \times 10$, then we have 10 depth slices, each of size 32×32. For the fully connected neural network, the whole layer would have $11 \times 11 \times 3 \times 32 \times 32 \times 10 = 3,717,120$ weights. However, for convolutional layer, since the neurons in the same depth slice share the parameters, it would only have $11 \times 11 \times 3 \times 10 = 3,630$ weights.

This procedure can also be seen as using filters to extract features from input volume. The learnable parameters (weights of neurons) are exactly the filter coefficients. Assume the input of convolutional layer is a three channel image, the first filter in the convolutional layer is of size $5 \times 5 \times 3$. The 5×5 (weight $\times$ height) is the kernel size of filter, which we will introduce in the following, the depth 3 is always decided by the convolutional layer input volume, and is totally the same as the full depth of the input volume.

During the forward propagation, the $5 \times 5 \times 3$ filter slides along the width and height of the input volume, from left to right, up to down. And the convolution between the filter coefficients and the input volume of the corresponding position is computed. With sliding, a $2-$dimensional activation map is produced. We can set some different filters in each convolutional layer, each of them produces a separate $2-$dimensional activation map. These maps are stacked along the depth dimension and produce the output volume. Therefore, the number of filters in convolutional layer is equal to the depth of convolutional layer and decides the output volume.

CNNs is actually a filter-based image processing system with self-learned filter coefficients. The following paragraphs will further introduce some unique concepts appeared in convolutional layers.

Kernel Size (K) The kernel size (K) is the size of filter extending along width and height, which is the most important hyperparameter for convolutional layer. The kernel

size decides the respective field of neuron in each depth slice. In practice, it always is set as an odd number, such as 3, 5, 7, 9 or 11.

Stride (S) Stride of a filter is the step the filter takes when it shifts over the input matrix. When the stride is 1, it means we move the filter one pixel at a time. When the stride is 2, the filter jumps 2 pixels at a time, which leads to the output size shrink. Stride is 3 or more is uncommon to see.

Padding (P) Sliding a filter around an input without padding would lead to the shrink of spatial size of the output, so it is common to pad zeros around the border of the input volume. The size of this zero-padding is a hyperparameter. Zero padding allows us to control the spatial size of the output volume. The most common condition is keeping the input and output width and height same by padding zeros around the input. The spatial size of the output volume can be computed by the given formula:

$$W' = \frac{(W - K + 2P)}{S} + 1, \tag{2.42}$$

where W' is the spatial size of the output volume, W is the spatial size of the input volume before padding, K is the kernel size of filter, P is the number of zeros padding on the border, S is the stride of filter.

Convolution Demo We can use a demo to demonstrate the behavior of a convolutional layer in detail. Assume the input x of the convolutional layer is an $3-$channel image with spatial size 5×5 (W=5), we pad 1 zero around the input volume (P=1) and get an input volume of size $7 \times 7 \times 3$. We express the three channels as $I_0 = x[:, :, 0]$, $I_1 = x[:, :, 1]$ and $I_2 = x[:, :, 2]$, respectively.

Then we further assume that there are two different filters (W_0 and W_1) in the convolutional layer, the kernel size of the filter is $K = 3$. Since the input has the depth of 3,

the filters are of size $3 \times 3 \times 3$. We use $w_{00} = w_0[:,:,0]$, $w_{01} = w_0[:,:,1], w_{02} = w_0[:,:,2]$, to denote the coefficients of every layer of filter W_0, and use $w_{10} = w_1[:,:,0]$, $w_{11} = w_1[:,:,1], w_{12} = w_1[:,:,2]$, to denote the coefficients of every layer of filter W_1.

We apply the filter with a stride of 2, then the spatial size of output volume o would be $W' = \frac{(W-K+2P)}{S} + 1 = 3$. Since we have two different filters in the convolutional layer, the output volume would be of size $3 \times 3 \times 2$. We use $O_0 = o[:,:,0]$ and $O_1 = o[:,:,1]$ to denote the two channels of outputs. Then the output O_0, O_1 can be calculated by:

$$O_0 = w_{00} * I_0 + w_{01} * I_1 + w_{02} * I_2 + b_0, \tag{2.43}$$

$$O_1 = w_{10} * I_0 + w_{11} * I_1 + w_{12} * I_2 + b_1, \tag{2.44}$$

An illustration of computing the first value of O_0 is shown in Figure 2.14.

Figure 2.15 shows the next step after computing the first value of the first output channel using filter W_0. We can image that other element in the output O_0 are computed in the same way by iteratively convolving the filter W_0 with the pixels in the sliding window (the window has a size same as kernel size and slides two pixels for every step) and adding with the bias b_0. Figure 2.16 shows the situation when we change the filter to W_1 and begin to produce the element in the second layer of output.

2.7.3 Non Linearity (ReLU)

As we introduced in neural network, ReLU stands for Rectified linear unit, which is a non-linear operation. In CNNs, ReLU layer follows with the convolutional layer, applies non-linearity to the output volume of convolutional layer. Figure 2.17 shows the ReLU operation on one channel of convolutional layer output volume.

Like in neural network, there are other non-linear operations such as sigmoid and tanh layers can be used instead of ReLU layer after convolutional layer. However, ReLU layer is the most common one in CNNs since its better performance.

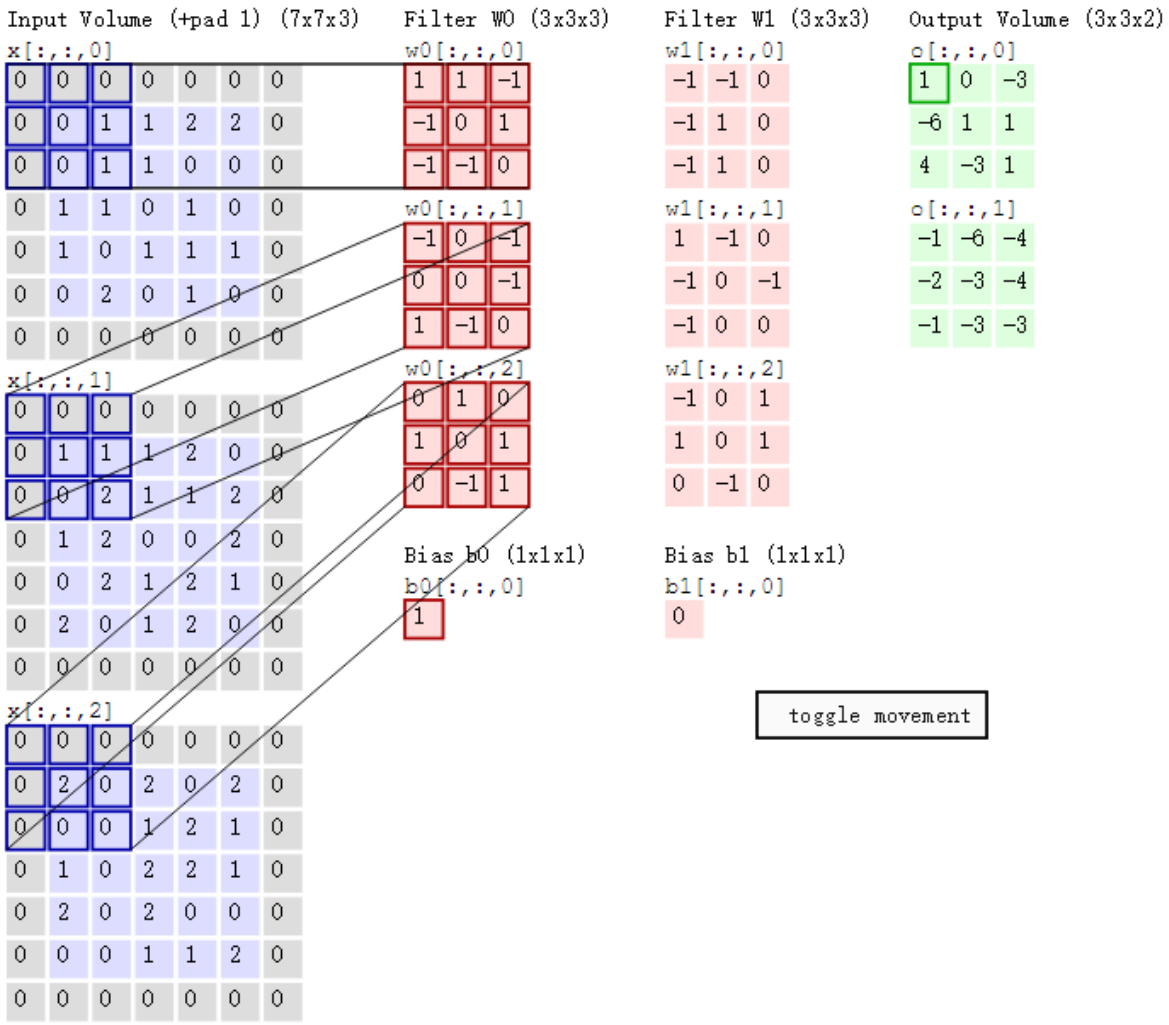


Figure 2.14: To produce the first value of O_0, the dot products between each layer of filter W_0 and the highlighted region of the same layer of input are calculated. Then the results are summed up, and offset by the bias b_0. The demo is cited from [29].

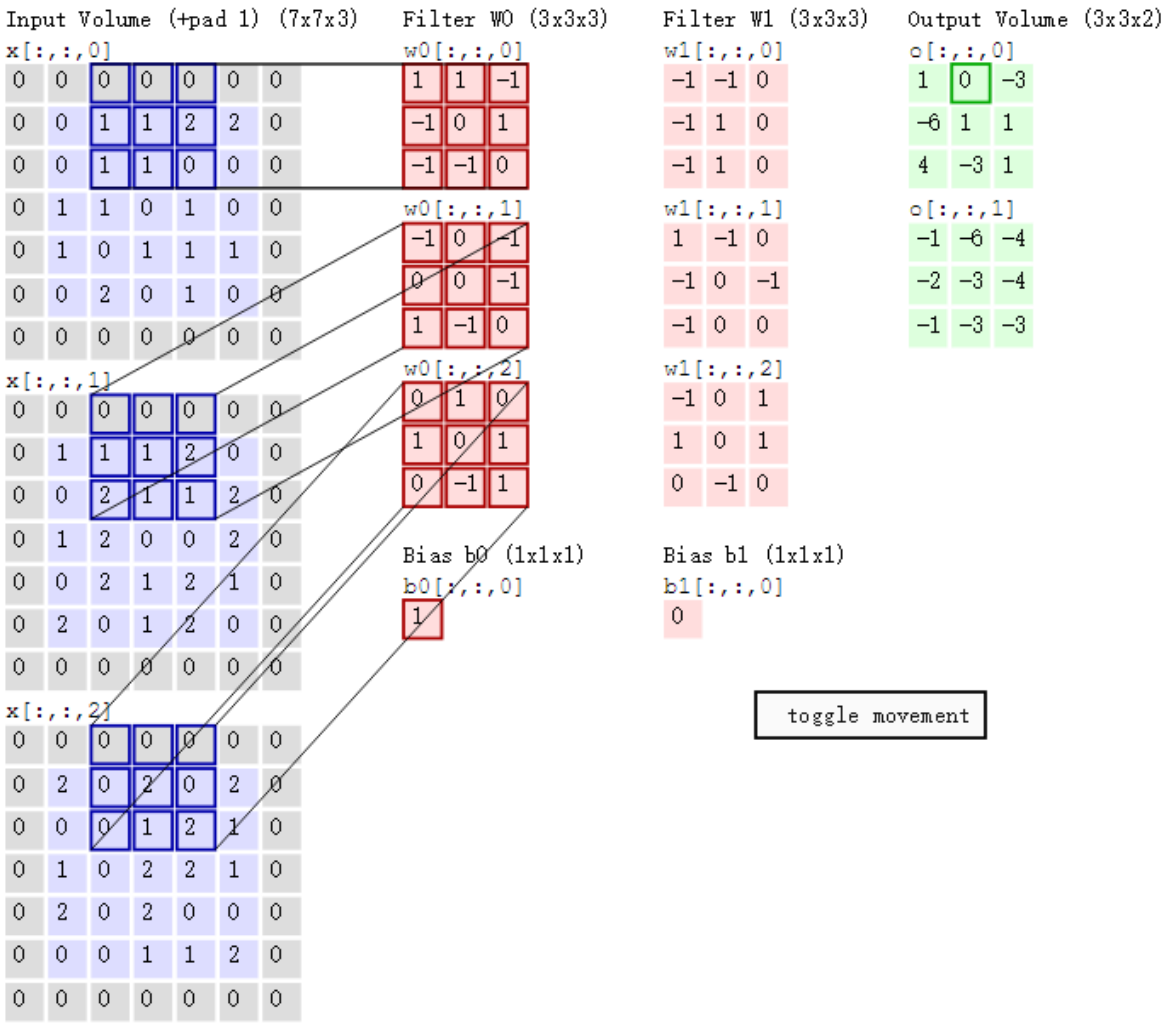


Figure 2.15: In the next step, since the stride is 2, the filter slides two pixels to the right on the input. The dot products between each layer of filter W_0 and new highlighted region of input are calculated, the results are also added with bias b_0 to produce the second value of O_0. The demo is cited from [29].

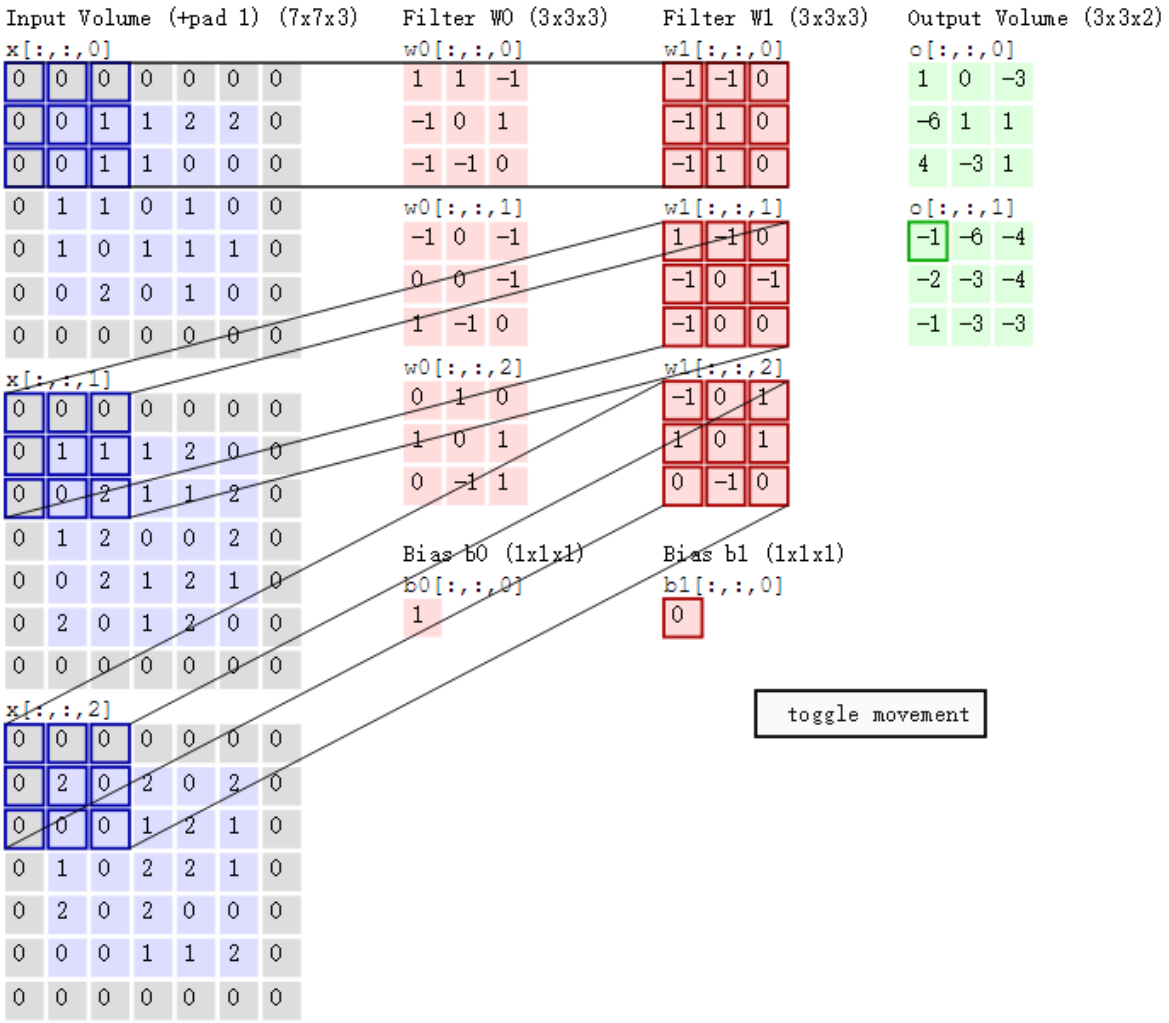


Figure 2.16: Producing the second channel of output by using filter W_1. The demo is cited from [29].

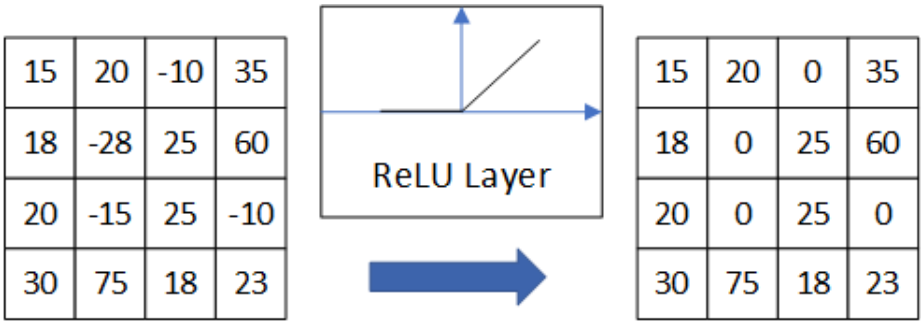


Figure 2.17: ReLU operation in CNNs.

2.7.4 Pooling Layer

Pooling layer is common to be inserted periodically in-between successive convolutional layers in CNNs. Pooling layers reduce the spatial size of the representations and reduce the number of parameters in the network. Hence, it effectively reduces the computation in CNNs and controls overfitting. It is also called subsampling or downsampling.

Except for reducing parameters, pooling layer also helps the representations learned by CNNs to be more invariant to small changes of the inputs. The invariance contains translation invariance, rotation invariance and scale invariance. It means that if the representations in the input of the pooling layer is shifted a few pixels, rotated a few degrees or changed in size, the output keep unchanged. Figure 2.18, Figure 2.19 and Figure 2.20 illustrates the translation invariance, rotation invariance and scale invariance with max pooling, respectively.

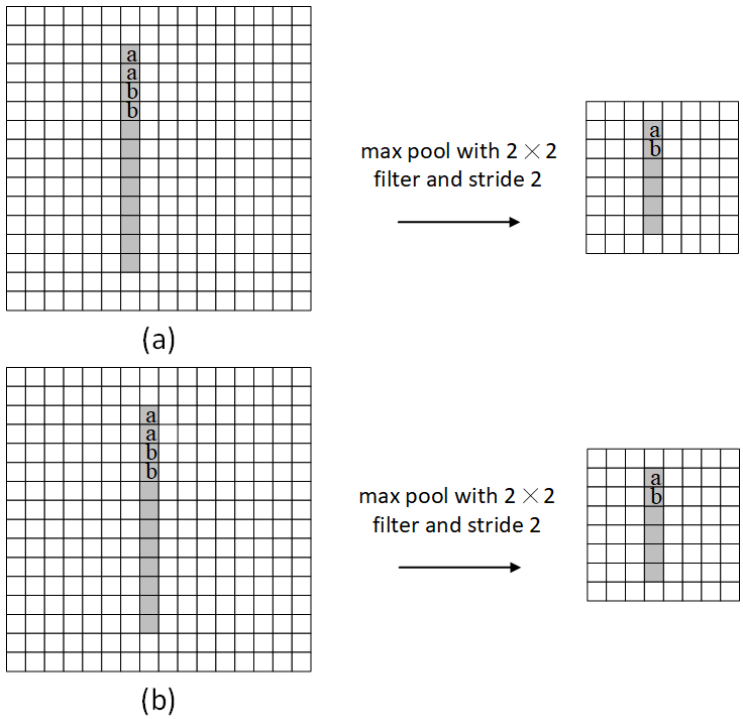


Figure 2.18: Translation invariance with max pooling layer.

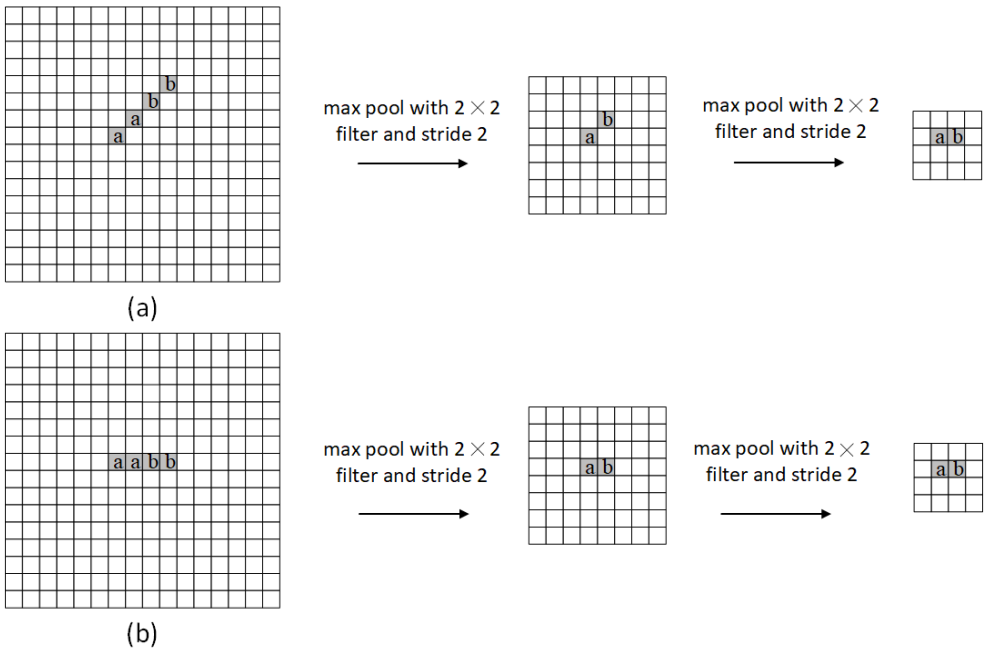


Figure 2.19: Rotation invariance with max pooling layer.

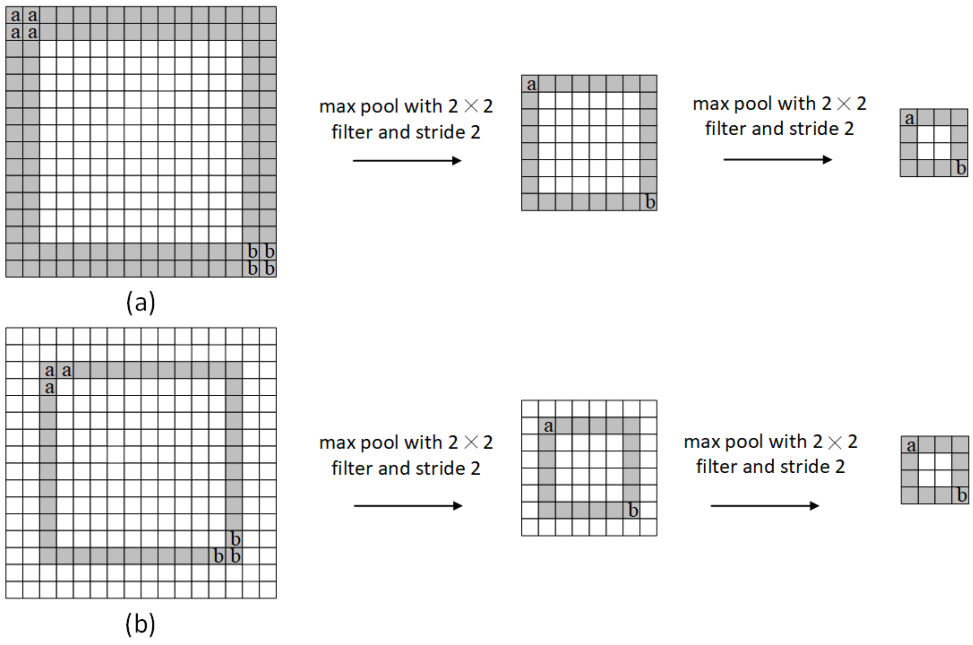


Figure 2.20: Scale invariance with max pooling layer.

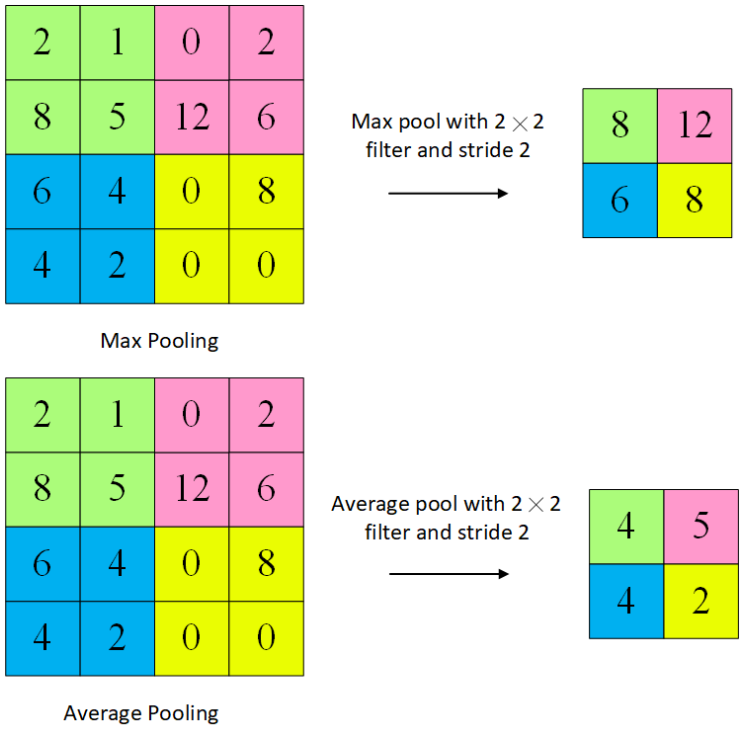


Figure 2.21: Pooling layers

We should notice that pooling operation is just a kind of downsampling without any parameters need to be learned. The concepts of kernel size, stride and padding in pooling layer is the same as those in convolutional layer. Figure 2.21 illustrates two kinds of common pooling layers: max pooling layer and average pooling layer. After the max pooling layer, only the maximum value in every kernel-size region is retained. For the average pooling layer, the average value of the kernel-size region is retained. Hence, the dimension of feature map shrinks after pooling layers in CNNs.

We should notice that the function of pooling layer can sometimes be replaced by convolutional layers with stride more than one. Since the learnable parameters in convolutional layer provide more flexibility, the convolutional layer with increased stride is considered to be better than pooling layer in recent works.

Chapter 3

Literature Review

As stated in Chapter 1, the alpha matting problem is a severely under-constrained problem. Although with the help of user interactions: trimap (dividing the image into three parts: user-defined foreground region, user-defined background region and unknown region) and scribbles (labeling known foreground/background region and unknown region with sparse scribbles), alpha matting is still a tough problem. In this chapter, we will introduce a number of alpha matting algorithms, which contribute to the development of alpha matting research.

3.1 Conventional Alpha Matting Algorithms

Before deep learning based alpha matting, solutions to the alpha matting problems are restricted to conventional algorithms. Those conventional algorithms can be divided into two categories: sampling-based methods and propagation-based methods. Both of them utilize strong correlation between neighboring pixels to alleviate the difficulties of solving the ill-posed matting problem.

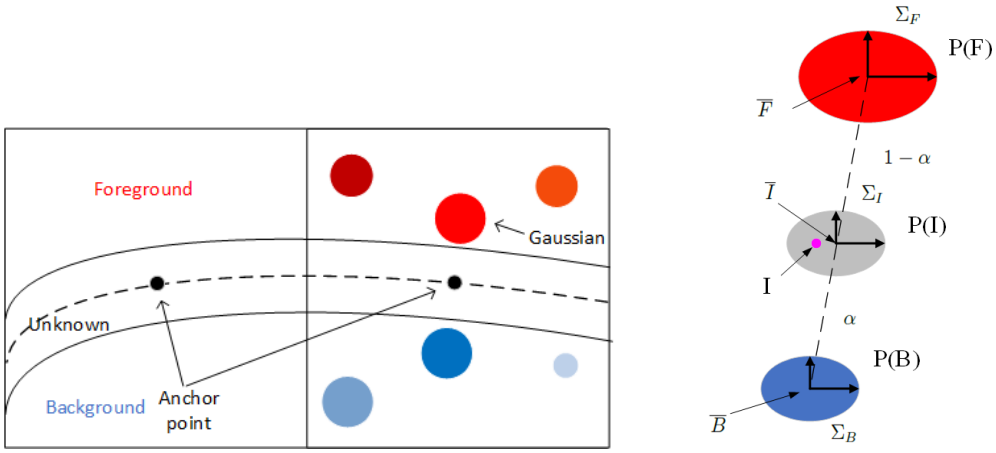
3.1.1 Sampling-based Methods

Sampling-based methods are based on a local spatial color coherence assumption, which is also called local color smoothness assumption. The assumption indicates that for a given unknown pixel i, the true foreground/background color F_i/B_i can be explicitly estimated by examining nearby pixels, which are located in the user-defined known foreground/background regions. After obtaining the candidate foreground and background colors of the unknown pixel, the alpha matte is calculated.

Sampling-based methods can be further categorized into two classes: parametric sampling methods and non-parametric sampling methods.

Parametric sampling methods compute alpha values based on the probability distributions of foreground and background colors. The representatives of this class are Ruzon and Tomasis work [49] and Bayesian Matting [8].

Ruzon and Tomasi's Matting Method In the Ruzon and Tomasi's matting method [49], the local sample region is a rectangular window centered on an "anchor point". As Figure 3.1(a) shows. Those known foreground pixels in the local window are divided into clusters and each cluster is fitted with a Gaussian distribution. Assume each of these Gaussians has mean $\overline{F}$ and covariance Σ_F. Then, both the foreground and the background obey a mixture of Gaussian distributions in color space. The observed color I is then modeled as coming from an intermediate distribution $P(I)$ between a pair of foreground and background distributions. Since there are a number of foreground and background pairs, the intermediate distribution is also a mixture of Gaussian distributions. The mean $\overline{I}$ and covariance Σ_I of each Gaussian are linearly interpolated values between the foreground and background Gaussian pair, according to the estimated α value. Figure 3.1(b) shows one pair of foreground and background distributions as an example. The optimal α value is the one that produces the distribution, where the observed color I has maximum probability.



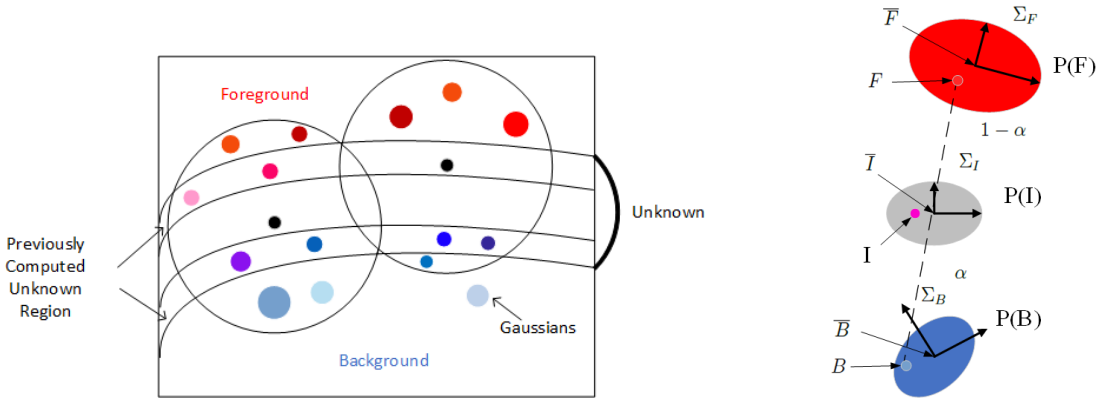
(a) Neighborhood determined by Ruzon's Method [49] (b) Alpha estimation model in Ruzon's Method [49]

Figure 3.1: Ruzon's Method [49] analyzes unknown pixels by local distributions. The rectangular window to the right of (a) is the local window, in which the known foreground and background colors are clustered and fitted with Gaussians. (b) shows how the α value of an unknown pixel is computed from a foreground and background Gaussian pair.

Bayesian Matting Method The Bayesian matting method [8] is based on the Ruzon and Tomasi's algorithm. It also clusters the colors of the known foreground and background regions, and fits them with mixtures of Gaussian distributions, but with some improvements.

- First of all, the Bayesian matting method defines the local sample region with a continuously sliding window instead of the rectangle window. The window slides inward from the foreground and background regions. Figure 3.2(a) shows this procedure.
- In addition, not only the samples in the known foreground and background region, the nearby unknown pixels, whose F, B, α values are estimated in the previous step, are also utilized for constructing Gaussian distributions.
- Furthermore, the contributions of nearby pixels are weighted by α and distance. The weighted mean colors $\overline{F}$, $\overline{B}$ and the weighted covariance matrixes Σ_F, Σ_B make the

color distribution models for foreground and background more robust.



(a) Neighborhood determined by Bayesian Method [8]

(b) Alpha estimation model in Bayesian Method [8]

Figure 3.2: Bayesian method [8]. (a) calculates the unknown pixels by analyzing the local color distributions in a sliding window. (b) shows the α value computed model.

After defining the neighborhood and the possibility distributions of nearby pixels, the matting problem is formulated in a Bayesian framework and solved by using the maximum a posterior (MAP) technique.

Although parametric sampling methods have been successful in dealing with some matting problems, the drawbacks are obvious, especially when the low-order parametric model is not feasible to the real color distributions and the correct foreground and background color samples are not in the local sample windows.

To overcome the drawbacks of parametric sampling methods, some non-parametric sampling methods have been proposed, which have the following four improvements:

- They don't fit probabilistic models to the foreground and background color distributions. On the contrary, they sample the colors in the user-defined foreground/background

region to represent the true foreground/background color F_i/B_i of the mixed unknown pixel i directly. Thus the quality of α computation mostly depends on whether the picked samples present the true foreground and background colors.

- Instead of sampling in the local window, more sampling strategies are proposed, as Figure 3.3 shows.

- In some matting methods, an objective function is built and the corresponding confidence values of sample pairs are calculated, in order to select the suitable sample pairs for unknown pixels [14] [21] [46] [52] [53] [61].

- For some algorithms, sampling is the first step. After obtaining a suitable initial α value, post-processing is performed to optimize that value. The post-processing method is minimizing an objective function, which contains data and smoothness terms, by constructing a Laplacian matrix L and solving a large linear equation [21] [46] [52] [53] [61]. The initial α value of unknown pixel with color I is estimated by the following equation:

$$\hat{\alpha} = \frac{(I - B^j)(F^i - B^j)}{\|(F^i - B^j)\|^2}, \tag{3.1}$$

where F^i and B^j are the foreground and background colors of a sampling pair with the sample indexes i and j.

(a) Image (b) RM [61] (c) SSM [14] (d) GSM [21] (e) CSM [53]

Figure 3.3: Sampling strategies of non-parametric sampling-based matting methods. The figure is cited from [28].

Robust Matting Method The Robust matting method (RM) [61] has two main innovations: applying a novel sampling strategy and using confidence value to evaluate the sample pairs in order to find "good" samples. The Robust matting method spreads the samples from the pixel with the shortest spatial distance to the unknown pixel, along the boundaries of known foreground and background regions, as shown in Figure 3.3(b).

For a foreground and background sample pair with color (F^i, B^j), the confidence value $f(F^i, B^j)$ is calculated by the following formulas:

$$R_d(F^i, B^j) = \frac{\|I - (\hat{\alpha}F^i + (1-\hat{\alpha})B^j)\|}{\|F^i - B^j\|}, \tag{3.2}$$

$$\begin{aligned} w(F^i) &= \exp\left\{-\|F^i - I\|^2 / {D_F}^2\right\} \\ w(B^j) &= \exp\left\{-\|B^j - I\|^2 / {D_B}^2\right\}, \end{aligned} \tag{3.3}$$

$$f(F^i, B^j) = \exp\left\{-\frac{R_d(F^i, B^j)^2 \cdot w(F^i) \cdot w(B^j)}{\sigma^2}\right\}. \tag{3.4}$$

Here, D_F and D_B are the minimum distance from unknown pixel to foreground and background samples in color space, respectively. σ is a tuning parameter, fixed to be 0.1.

The confidence value is large if 1) the observed color I fits Equ. (1.1) well, 2) the sample pair (F^i, B^j) is widely separated in color space, and 3) the samples are not similar to color I. We should notice that all the criteria for sample selection are based on the color space information, no other information is under consideration.

After examining the confidence value of every pair in the large number of candidates, the samples with high confidence are used in the following optimization process. The alpha matte optimization process can be seen as a graph-based image labelling problem, in which the matting energy function with data term and neighborhood smoothness term is minimized by a random walk [19].

Improved Color Matting Method The improved color matting method [46] is inspired by the robust matting method, but with improvements in two aspects: sampling strategy and confidence values calculation.

In the sampling step, the improved color matting method [46] assumes the foreground object is spatially connected. It spreads the samples from the pixel with the shortest geodesic distance to the unknown pixel along the boundaries of known foreground region. Figure 3.4 illustrates the difference between the robust matting and the improved color matting. The blue nodes are the foreground samples defined by the robust matting and the yellow ones are foreground samples collected by the improved color matting. As we can see, the foreground color of the unknown pixel (green point) is dark brown, only the improved color matting method collects the right foreground candidates. Hence, sampling based on the geodesic distance is better than the spatial distance in some situations.

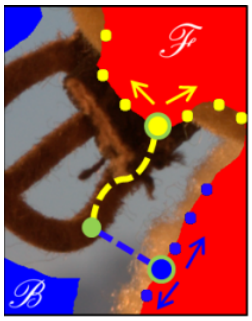


Figure 3.4: The comparison of two methods in spreading foreground samples. The figure is cited from [46].

In calculating confidence values of samples, the author obeys the first two of the three criteria proposed by the robust matting algorithm [61]: 1) the colors of foreground and background samples F^i, B^j and the observed color I fit the linear model in Equ. (1.1), 2) F^i and B^j are widely separated in color space, and the author does not follow the third

criterion by changing the weights $w(F^i)$ and $w(B^j)$ as:

$$\begin{aligned} w(F^i) &= \exp\left\{\frac{-\max_i(\|F^i - I\|^2)}{\|F^i - I\|^2}\right\} \\ w(B^j) &= \exp\left\{\frac{-\max_j(\|B^j - I\|^2)}{\|B^j - I\|^2}\right\}. \end{aligned} \tag{3.5}$$

Then the confidence value f for each sample pair is computed by Equ. (3.4). The sample pair with the highest confidence value $\hat{f}$ is selected. In this method, the color space information is also the only criterion considered in sample selection.

The alpha matte is also optimized by minimizing a cost function with data term and smoothness term. The data term is formed by combining the pixel-wise estimated $\hat{\alpha}$ with its confidence $\hat{f}$. And the smoothness term consists of a matting Laplacian matrix L of [34]:

$$\alpha = \arg\min \alpha^\top L\alpha + (\alpha - \hat{\alpha})^\top \hat{\Gamma}(\alpha - \hat{\alpha}). \tag{3.6}$$

Here $\hat{\alpha}$ is the initially estimated alpha value. $\hat{\Gamma}$ is a diagonal matrix, which trades off the weight between two terms. The diagonal element $\hat{\gamma_i}$ of $\hat{\Gamma}$ satisfies: $\hat{\gamma_i} = \gamma\hat{f_i}$, where $\gamma = 0.001$.

The improved color matting method performs better than the robust matting, but it faces a problem when dealing with transparent object abstraction. And sampling based on the shortest geodesic distance may also lead to missing of suitable samples.

Shared Sampling Matting Method The shared sampling matting method (SSM) [14] believes that pixels in a small neighborhood always have similar α, F and B values. Hence, the share sampling matting method shares the computation results with adjacent pixels.

As shown in Figure 3.3(c) and Figure 3.5, in sampling stage, the unknown pixel p is defined as an origin, a number of rays emit along with different directions from p. The rays divide the image plane into equal areas. Only the nearest foreground/background

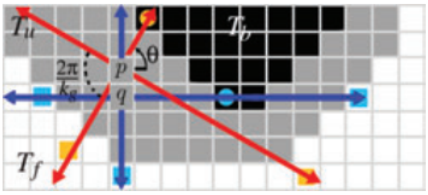


Figure 3.5: Shared Sampling Matting. The figure is cited from [14].

samples on each ray would be selected as candidates of p. These samples are always along the boundaries of user-defined foreground and background regions in a trimap.

For any unknown pixel p in a 3×3 neighborhood, the angle between the first ray and x-axis is defined as θ ($\theta \in [0, \frac{\pi}{2}]$). Different unknown pixels in that 3×3 window have different θ values. This ensures that the rays of neighboring pixels have a different orientation and pass through different regions to sample variant pixels.

After having the candidates, a new objective function, taking the color fitness, spatial distance, and probabilistic information into consideration, is used to select the best sample pair of each unknown pixel. Then the best sample pair of p and the best sample pairs of its k_r nearest unknown pixels construct a sample set. Three best sample pairs in the sample set are used to calculate α of unknown pixel p.

Although the sampling range of this method is enlarged and the correlation of neighborhood is utilized, the true colors are still likely to be missed since only samples along the boundary are taken into account and the number of samples are limited.

Global Sampling Matting Method To enlarge the sampling range and avoid missing true colors, He et al. proposed a global sampling matting method (GSM) [21], which samples all the known foreground and background pixels along the boundary of the unknown region. As shown in Figure 3.3(d).

An objective function, taking the color fitness and spatial distance into consideration,

is proposed to select the best sample pair of unknown pixel from the big sample set. And the initially alpha value $\hat{\alpha}$ for each pixel is estimated through Equ. (3.1).

Finally, the alpha matte is optimized by minimizing a global energy function with data term and smoothness term:

$$\alpha = \arg\min \alpha^\top L\alpha + \lambda(\alpha - \hat{\alpha})^\top G(\alpha - \hat{\alpha}), \tag{3.7}$$

where λ is a weight to trade off two terms, $\hat{\alpha}$ is the initially estimated alpha value, L is a matting Laplacian matrix proposed by [34]. G is a diagonal matrix, whose diagonal element is a constant for known pixel, and a confidence g for unknown pixel, where $g = \exp(-\frac{f_c(F^i,B^j)}{2\sigma^2}), (\sigma = 1)$.

The global sampling matting method gathers more samples, even including the pixels far away from the unknown one, and also utilizes the correlation among neighboring pixels. However, the limitation of sampling range (just along the boundary of the trimap) and sample selection criteria (color fitness and spatial distance) may also lead some errors in the final alpha matte.

Comprehensive Sampling Matting Method The comprehensive sampling matting method (CSM) [53] constructs a comprehensive sample set, which contains samples: not only along the boundary of the trimap, but also inside the user-defined known foreground and background regions. It tries to ensure that samples from each color distribution are contained, as Figure 3.3(e) shows.

The objective function proposed by the comprehensive sampling matting method contains three terms: color fitness, spatial distance and color statistics of the image. The color fitness term is similar with the one in [14] [21] [46] [61], which detects if the observed color of the unknown pixel is a linear combination of the colors of F^i and B^j. The spatial distance term tests the spatial distance between the sample pairs and the unknown pixel,

and prefers the spatially close pairs. The last term favors the sample pairs coming from well separated color distributions.

After using the object function to select the best sample pair, the initial alpha value of the unknown pixel: $\hat{\alpha}$ is calculated by Equ. (3.1). Finally, the comprehensive sampling matting method optimizes the alpha values by minimizing a cost function with a data term and a smoothness term. The data term combines the pixel-wise initial $\hat{\alpha}$ with its confidence value $\hat{f}$ (the result of the former objective function). The smoothness term contains a matting Laplacian matrix L [34]. The complete cost function is defined as:

$$\alpha = \arg\min \alpha^\top L\alpha + \lambda(\alpha - \hat{\alpha})^\top D(\alpha - \hat{\alpha}) + \gamma(\alpha - \hat{\alpha})^\top \hat{\Gamma}(\alpha - \hat{\alpha}), \tag{3.8}$$

where D is a diagonal matrix, whose diagonal element is 1 for known pixels and 0 for unknown ones. $\hat{\Gamma}$ is also a diagonal matrix, whose diagonal element is 0 for known pixels and $\hat{f}$ for unknown ones. λ and γ are weighting parameters. $\hat{\alpha}$ is the initially estimated alpha value.

Weighted Color and Texture Matting Method The aforementioned sampling methods explore different kinds of sampling strategies, but only consider color information when sampling. The weighted color and texture matting method [52] selects the sample candidates relying on both the color information and the texture information.

The specific sampling method is similar with the shared sampling matting method [14]. An objective function, detecting the color fitness and the compatibility of color and texture feature, is proposed to select the best sample pairs from the sample set. After calculating the initial alpha matte, the weighted color and texture matting method smooths the alpha matte with the similar post-processing method proposed by the comprehensive sampling matting method [53].

As various sampling strategies and sampling selection criteria have been proposed, some recent sampling-based methods have achieved impressive results. However, when the

image content is complex, the trimap is coarse, or the suitable sample pairs are failed to be collected due to the limitation of sampling range, the quality of the matte will attenuate. To avoid the drawbacks, caused by imperfect sampling, the propagation based matting methods are proposed.

3.1.2 Propagation-based Methods

The propagation-based methods do not calculate the alpha value by estimating the foreground and background colors of the unknown pixel. Instead, it defines an affinity matrix representing the similarity of neighboring pixels (i.e. sharing the same α) and propagates the alpha values of known region into the unknown ones. Some representative propagation-based matting methods are described in the following.

Poisson Matting Method The Poisson matting method [57] is proposed based on the assumption that the colors of neighboring foreground/background pixels change smoothly. It takes the partial derivatives (∇) on both sides of Equ. (1.1), neglects the gradients of F and B, which are much smaller than the gradient of α, and gets the approximate matte gradient from the input image:

$$\nabla\alpha_i \approx \frac{1}{(F_i - B_i)}\nabla I_i. \tag{3.9}$$

Then the matte can be calculated by solving the associated Poission equation:

$$\Delta\alpha \approx \mathrm{div}\Big(\frac{\nabla I}{F - B}\Big), \tag{3.10}$$

where Δ is the Laplacian operator and div is the Divergence operator, according to the Dirichlet boundary condition, which is defined by the trimap:

$$\alpha_j = \begin{cases} 1 & \text{if pixel } j \text{ belongs to foreground pixels,} \\ 0 & \text{if pixel } j \text{ belongs to background pixels.} \end{cases} \tag{3.11}$$

The Poission equation is solved by Gauss-Seidel iteration with over-relaxation method. Since the initial values of F and B are selected from the nearest samples in the foreground and background regions, and no effective information are provided to optimize F and B in the iteration process, the drawbacks included in sampling-based methods may also be preserved in this method. The Poission matting method also applies some local refine operations, such as channel selection and filtering to allow user to manipulate the gradient field of alpha matte locally. However, such operations are usually time-consuming in practice.

Random Walk Matting Method The random walk matting method [19] was proposed in 2005, after Poisson matting. Given a trimap, the random walk matting method defines the α value of each unknown pixel as the probability, that a random walker starts from this location and arrives at a foreground pixel before a background one. To guide the trajectory of the random walker, a weighted graph with nodes and edges is defined. Each pixel is treated as a node, connecting with its four neighboring pixels. The weight of the edge between pixel i and its neighborhood j indicates the affinity of these two pixels (i.e. sharing the same α), and can be defined as:

$$w_{ij} = \exp\Big(\frac{\|I_i - I_j\|^2}{\sigma^2}\Big), \tag{3.12}$$

where I_i is a vector representing the RGB color of pixel i, σ is a free parameter, set as $\frac{1}{30}$ in [19]. The probability that a random walker at i moves to j can be then computed by: $p_{ij} = \frac{w_{ij}}{d_i}$, where d_i can be computed as: $d_i = \sum_j w_{ij}$, the sum of the four weights of edges that connect to pixel i. To calculate the color distance in Equ. (3.12) better, a Locality Preserving Projections (LPP) technique is used. It projects the color image from RGB channels to LPP channels, so as to distinguish the object boundary better. The definition of weights of edges are changed accordingly, based on the application of LPP algorithm:

$$w_{ij} = \exp\Big(\frac{(I_i - I_j)^\top Q^\top Q(I_i - I_j)}{\sigma^2}\Big), \tag{3.13}$$

where Q is the solution of a generalized eigenvector problem:

$$ILI^\top x = \lambda IDI^\top x, \tag{3.14}$$

where I is a $3 \times N$ matrix, whose column is ${I_i}^\top$. N is the number of pixels. D is a $N \times N$ diagonal matrix, whose diagonal element $D_{ii} = d_i$. L is the $N \times N$ graph Laplacian matrix formulated as:

$$L_{p_i p_j} = \begin{cases} d_i & \text{if } i = j, \\ -w_{ij} & \text{if } p_i \text{ and } p_j \text{ are adjacent nodes,} \\ 0 & \text{otherwise,} \end{cases} \tag{3.15}$$

where $L_{p_i p_j}$ indicated that the matrix L is indexed by pixels p_i and p_j.

It turns out that the desired random walker probabilities (alpha matte) are the global minimum to the Dirichlet energy function, with the boundary conditions (defined in Equ. (3.11)):

$$\alpha = \arg\min \alpha^\top L \alpha, \tag{3.16}$$

where L is the graph Laplacian matrix. The alpha matte will be calculated by solving the system with linear equations. The random walk matting method makes full use of local neighboring pixel correlation and produces a smooth alpha matte. However, sometime, the final alpha matte is over-smoothed and some pure foreground and background pixels are estimated as mixed ones by mistake.

Closed-form Matting Method Compared with the Poission matting algorithm that believes the colors of neighboring pixels change smoothly, the Closed-form algorithm [34] obeys the color line assumption. The color line assumption means that in a small window (3×3 or 5×5) around a pixel i, the foreground (or background) color of each pixel in this local window lies on a straight line in the color space. The color line assumption would be invalid if the window is too big. If the color line assumption holds, the alpha value of the pixel i in the small window w of a color image satisfies the given function:

$$\alpha_i = a^\top I_i + b, \ \forall i \in w, \tag{3.17}$$

where a is a 3×1 vector and b is a number. For every small window, a and b are fixed.

I_i is also a 3×1 vector, representing the three color channels of the pixel i.

The Closed-form algorithm derives the alpha matte by minimizing the following cost function:

$$J(\alpha, a, b) = \sum_{j=1}^{N} \Big(\sum_{i \in w_j} \big(\alpha_i - (a_j^\top I_i + b_j)\big)^2 + \varepsilon \|a_j\|^2 \Big), \tag{3.18}$$

where N is the number of pixels, w_j is the small window centered at pixel j, ε is a regularization parameter. Since the windows overlap between each other, the information between neighboring pixels can be propagated. By minimizing the cost function w.r.t. (a, b), the cost function becomes quadratic in α:

$$J(\alpha) = \alpha^\top L \alpha. \tag{3.19}$$

Here, α is an $N \times 1$ vector, where N is the number of unknown pixels. L is also a graph Laplacian matrix, with size $N \times N$. The (i, j)-th element of L can be formulated as:

$$\sum_{k|(i,j) \in w_k} \left(\delta_{ij} - \frac{1}{|w_k|} \Big(1 + (I_i - \mu_k) \big(\Sigma_k + \frac{\varepsilon}{|w_k|} I_3 \big)^{-1} (I_j - \mu_k) \Big) \right), \tag{3.20}$$

where δ_{ij} is the Kronecker delta, $|w_k|$ is the number of pixels in w_k, μ_k and Σ_k are the mean vector and covariance matrix of the colors in window w_k, I_3 is a 3×3 identity matrix. To get the alpha matte, which is consistent with the user scribbles, they solve the following function:

$$\alpha = \arg\min \alpha^\top L \alpha, \text{ s.t. } \alpha_i = s_i \ \forall i \in S. \tag{3.21}$$

Here, S is the set of scribbled pixels, s_i is the corresponding alpha value indicated by the scribbles (0 or 1).

We should notice that the Laplacian matrix L also can be written as $L = D - W$, where W is a symmetric weight matrix, whose off-diagonal elements are defined by:

$$W(i, j) = \sum_{k|(i,j) \in w_k} \frac{1}{|w_k|} \Big(1 + (I_i - \mu_k) \big(\Sigma_k + \frac{\varepsilon}{|w_k|} I_3 \big)^{-1} (I_j - \mu_k) \Big). \tag{3.22}$$

D is also a diagonal matrix $D(i, i) = \sum_j W(i, j)$, which is the same as [19]. Hence, the matrix L in the Random walk matting method [19] and the one in the Closed-form

matting method [34] are all the graph Laplacian matrices used in spectral methods for segmentation. The difference between these two methods is the different affinity function ($W_{i,j}$) for neighboring pixels. We call these kinds of propagation-based matting algorithms as matting-Laplacian-based algorithms. In mathematics, they derive the alpha matte by solving a sparse linear system.

The Closed-form matting is also often used as a smooth term in post-process after sampling [21] [46] [52] [53] [61]. This method can work well if the color line model is satisfied in a small window, but may cause over-smoothing in complex regions of image. In the meantime, it may consume a lot of time to solve the large linear system.

KNN Matting Method The KNN matting method [6] obeys the assumption of non-local principle, which is first proposed by [33]. The nonlocal principle indicates that the alpha value α_i of a pixel i can be expressed as a weighted sum of the alpha value α_j. j is the nonlocal neighboring pixel of i, which has similar appearance with i. Based on this assumption, the KNN matting obtains nonlocal neighboring pixels j of i by collecting the first K nearest neighbors of pixel i in the high dimensional feature space. The KNN defines a novel feature vector $X(i)$ for pixel i:

$$X(i) = (\cos(H_i), \sin(H_i), S_i, V_i, x_i, y_i), \tag{3.23}$$

where H_i, S_i and V_i are the hue, saturation and lightness values in HSV color space and (x_i, y_i) is the spatial coordinate of pixel i.

Given the feature vector for each pixel, a kernel function, which formulates the affinity between two pixels, is defines as:

$$k(i,j) = 1 - \frac{\|X(i) - X(j)\|}{U}, \tag{3.24}$$

where U is the least upper bound of $\|X(i) - X(j)\|$ to squash $k(i,j)$ into $[0,1]$. $\|\cdot\|$ is L_1 norm, calculating the sum of the absolute values of the difference between the two vectors.

The matting Laplacian matrix L can be obtained by:

$$L = D - W, \tag{3.25}$$

where W is an $N \times N$ affinity matrix with the $(i, j)-$th element defined as $k(i, j)$. D is an $N \times N$ diagonal matrix, $D_{ii} = \sum_j k(i, j)$. N is the number of pixels.

Finally, combining the user-constraint information, the alpha matte is the closed form solution of the equation:

$$(L + \lambda Q) \sum_{i}^{n} \boldsymbol{\alpha}_i = \lambda \mathrm{m}, \tag{3.26}$$

where $Q = \mathrm{diag(m)}$ and m is a binary vector of indices of all the user-constraint pixels, λ is a parameter. The equation can be solved by the preconditioned conjugate gradient method.

Comparing with the Closed-form matting method [34], the KNN matting method performs well in dealing with highly textured regions or regions with holes in the trimap. However, when encountering fur regions, such as animals' hairs, its performance is lack of smoothness.

In summary, the propagation-based methods utilize the relations among neighboring pixels and propagate alpha values from the known regions to the unknown ones. They perform well when the underlying assumptions stand. However, when the assumption is violated, strong edges existed in highly textured regions block the propagation, and the foreground and background colors are extremely similar, the matting quality deteriorates. In addition, the propagation process may accumulate the errors. Although some recent sampling-based methods tend to utilize the propagation-based methods in their post-processing so as to optimize their matting results, these kinds of combinations are more like to be a simple tradeoff between the sample colors and pixel affinities, rather than a perfectly complementary. As a result, both of them may fail to deal with some complicated scenes with dense textures or color fusion.

3.2 Deep Learning based Alpha Matting Algorithms

To solve the problems encountered by conventional matting algorithms, the matting methods based on deep learning are proposed following the explosion of deep learning technique researching.

DCNN Matting Method The deep Convolutional Neural Network (DCNN) matting method [7] can be regarded as the first matting method that applies the deep Convolutional Neural Network, a branch of deep learning, in solving natural image matting problem.

The training dataset contains more than a hundred thousand of image patches, which are original from 27 training images. With compositing the foreground onto different backgrounds and data augmentation strategies, those training patches are generated. The testing dataset contains 8 testing images. Both of 27 training and 8 testing images are provided by *alphamatting.com* website [47].

The CNN architecture is an alternative pile of convolutional layer and ReLU layer, the number and the size of kernels for each convolutional layer is illustrated in Figure 3.6. In the training stage, the input of the CNN model is a 5-channel image patches ($27 \times 27 \times 5$), which combines the alpha mattes obtained by local (Closed-form matting [34]) and nonlocal (KNN matting [6]) propagation-based matting algorithms with the corresponding normalized RGB color image. The output is a predicted alpha matte patch ($15 \times 15 \times 1$), which compares with the ground truth alpha matte with the Euclidean loss function. The errors are back propagated to the CNN model, from the output layer to the input one, in order to update the kernels' weights for each layer. In the testing stage, the input of the CNN model is the original resolution of the testing images and after a single forward pass, an estimated alpha matte with equal size is obtained.

Although the combination of the matting results from the previous conventional matting methods and the deep CNN technique leads to a better alpha matte at that time,

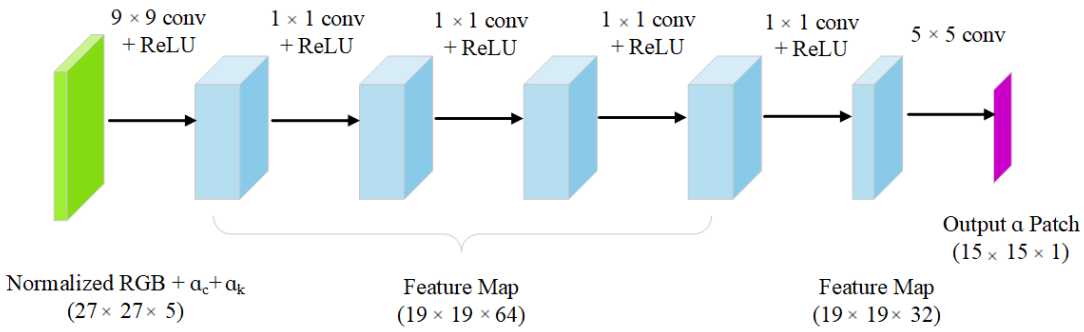


Figure 3.6: The deep CNN architecture of the DCNN matting method [7].

there are some drawbacks in the DCNN matting method. Firstly, getting the alpha mattes from other conventional matting methods before training the network is time consuming. And adding more matting results from conventional methods as input can really reach a better result, but it is considered to be kind of cheating, because the approximate answers have been shown to the network when testing. Secondly, the usage of the Close-form and the KNN matting's results makes this method preserve the matting errors existed in the previous matting methods. Thirdly, the dataset is so small, only containing 27 training images and 8 testing images, and most of them are indoor scenes. The constraints of the dataset may lead to the overfitting when training the network and may bias the network to be more incentivized to fit to this kind of dataset instead of the real scenes.

Automatic Portrait Matting Method The automatic portrait matting method [55] pays attention to calculating the alpha matte for the portrait images. Their system includes two modules: trimap labelling and image matting. In the trimap labelling module, a portrait image and its pre-trained shape mask is given as input, then a convolutional neural networks (CNNs) is used to produce three maps: F^s, B^s and U^s, which indicates the probability that each pixel belongs to the foreground, background or unknown regions respectively. Then, a softmax function is used to transform the three channels into the probability maps F and B, which are the input of the image matting module.

In the image matting module, the alpha matte is computed through a propagation-based matting method, which can be expressed as:

$$\alpha = \arg\min \lambda \boldsymbol{\alpha}^{\top} \mathbf{B} \boldsymbol{\alpha} + \lambda (\boldsymbol{\alpha} - \mathrm{C})^{\top} \mathbf{F} (\boldsymbol{\alpha} - \mathrm{C}) + \boldsymbol{\alpha}^{\top} L \boldsymbol{\alpha}, \tag{3.27}$$

where $\boldsymbol{\alpha}$ is the alpha matte vector, C is an all-1 vector, $\mathbf{B} = \mathrm{diag}B$, $\mathbf{F} = \mathrm{diag}F$, and L is the matting Laplacian matrix proposed by [34], λ is an adjusting parameter. According to the solution of Equ. (3.27), the image matting module can be formulated as:

$$f(F, B; \lambda) = \lambda (\lambda \mathbf{B} + \lambda \mathbf{F} + L)^{-1} F, \tag{3.28}$$

where F and B are the input probability maps and λ is the parameter need to be learnt. $f(F, B; \lambda) = \boldsymbol{\alpha}$ defines the forward propagation. Then a loss function $L(\boldsymbol{\alpha}, \boldsymbol{\alpha}_{gt})$ is defined to measure the errors between the predicted alpha matte and the ground truth alpha matte. With back propagating the errors, the parameters in the CNNs are updated.

Although this matting method generates a trimap with CNNs automatically, it is still not a complete matting method based on deep learning. Because CNNs is only used to generate the probability maps and the truly matting process is still based on the propagation-based matting method. The errors produced in the propagation-based matting algorithm will still affect the matting results. In addition, this matting method just pays attention to portrait images. We still hope to find a reasonable matting method based on deep learning that can estimate alpha matte directly from the original image, and can deal with all the natural images, not only for the portrait images.

Deep Image Matting Method The deep image matting method [65] is the first matting method that uses a complete CNNs to learn the alpha matte directly, given an image and its corresponding trimap as input. Its network has two stages: encoder-decoder stage and matting refinement stage. In the encoder-decoder stage, the input is a 3-channel image patch and its trimap. The loss function is the weighted sum of alpha prediction loss and the compositional loss. The alpha prediction loss is the absolute difference between

the predicted alpha matte and the ground truth alpha matte. The compositional loss is the absolute difference between the ground truth RGB color image and the predicted RGB color image, which is composited by the ground truth foreground, ground truth background and the predicted alpha matte. The output of this stage is the predicted alpha matte. In the refinement stage, the predicted alpha matte from first stage and the original 3-channel image patch are concatenated as input. Then the input is fed to a few convolutional layers, the output of these layers is then added with the predicted alpha matte produced by the encoder-decoder stage to produce the final alpha matte. In fact, the refinement stage is worked as a residual block [24]. The loss function of this stage is only the alpha prediction loss. The whole network is shown in Figure 3.7.

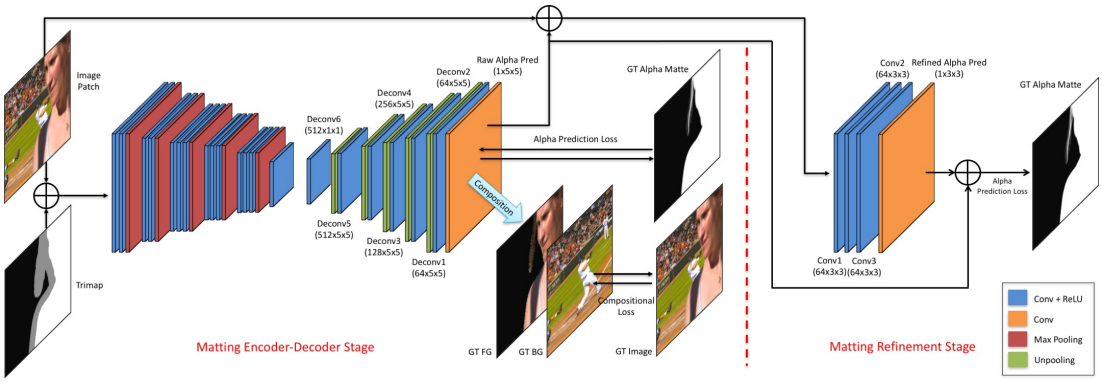


Figure 3.7: The CNNs architecture of the deep image matting method. The figure is cited from [65].

The deep learning method achieves a state-of-the-art performance. Meanwhile, it proposes a new large matting dataset, which improves the possibility of solving matting problem through deep learning technique. The only imperfection of this method is the complicated network architecture and the huge number of parameters. Without the refinement stage, the network has generated nearly 26 million parameters. Training such a complicated network leads to a huge time consumption. In the meanwhile, the network is lack of flexibility, if the dataset is enlarged in the future, this network architecture will not fit to that huge dataset anymore. Our proposed matting method is also based on an end-to-end

CNNs, but will be more flexible and simple, without sacrificing the quality of matting results.

GAN Matting Method The GAN matting method [38] is a novel alpha matting method based on deep learning, it can be seen as the first matting method using a generative adversarial networks (GANs) to solve matting problem for natural images. The generative adversarial networks (GANs) has two parts: the generator G and the discriminator D.

The input of G is a 4-channel image, composited by the RGB color image and the corresponding trimap. G aims at predicting the right alpha matte, which is similar to the ground truth. D tries to distinguish the differences between the real 4-channel inputs and the fake 4-channel inputs. The real 4-channel input is composited from the foreground, ground truth alpha matte, the background and the trimap. And the fake one uses the predicted alpha matte to replace the ground truth alpha matte in the real 4-channel input.

The generator of this network is a convolutional encoder-decoder network. For the encoder, the ResNet50 [24] architecture is pre-trained on Imagenet [48], and modified appropriately as the first part. Then the atrous spatial pyramid pooling (ASPP) module from [5] is added to output $256 \times 40 \times 40$ feature maps. For the decoder, bilinear interpolation, unpooling and fractionally-strided convolution are used to upsample the feature maps back to 320×320 (the size of input image patch). And some skip lines connecting the encoder layers to the decoder layers are also added so as to retain some local information of inputs. The generator network architecture is illustrated in Figure 3.8.

The discriminator of this network is a PatchGAN proposed by [26], which tries to judge every $N \times N$ patch of the input image as a real one or a fake one. The input of D has 4-channels: the original RGB color image and the trimap.

The loss function of this network contains three parts: the alpha prediction loss L_{alpha},

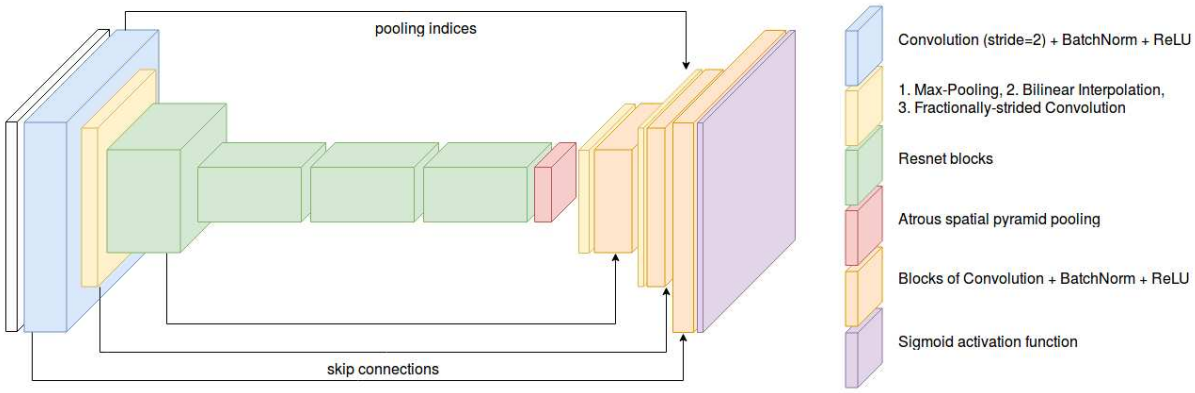


Figure 3.8: The architecture of the generator network of the GAN matting method. The figure is cited from [38].

the compositional loss L_{comp}, and the adversarial loss L_{GAN}. The former two are proposed by [65], and the third one is a specific one for GAN matting, which is formulated as:

$$L_{GAN}(G, D) = \log D(x) + \log(1 - D(C(G(X)))), \tag{3.29}$$

where x is the real 4-channel input (with ground truth alpha matte). $C(y)$ is the composition function, compositing the predicted alpha matte from G with the foreground, background and trimap to form the fake image. G tries to minimize L_{GAN} by generating alpha matte similar to ground truth, while D attempts to maximize it by telling real from fake composited images. Consequently, the alpha matte is obtained by:

$$\alpha = \arg\min_{G}\max_{D} L_{AlphaGAN}(G, D), \tag{3.30}$$

where

$$L_{AlphaGAN}(G, D) = L_{alpha}(G) + L_{comp}(G) + L_{GAN}(G, D). \tag{3.31}$$

The GAN matting method achieves a reasonable performance, but the performance is not good as ours. In our opinion, GAN is not suitable for solving matting problems.

GAN is a deep learning, unsupervised learning technique proposed by Ian Goodfellow. In GAN, we have a generator and an adversarial network called discriminator. Hence the

network is named as generative adversarial network. The objective of generator is to model or generate data that is very similar to the training data. In other words, generator needs to generate data that is indistinguishable from the real data. Generated data should be such that discriminator is tricked to identify it as real data. Meanwhile, the object of discriminator is to identify if the data is real or fake. The discriminator gets two sets of input. One input comes from the training dataset and the other input is the modelled dataset generated by generator. Generator can be thought as counterfeiters making fake items which look exactly like real items. Discriminators can be considered as law-executors trying to detect the counterfeit items. Counterfeiters and law-executors both are trying to beat each other at their game. In general, if we use GAN to deal with images, GAN is more suitable to tell the "fake" from "real" visually. In other words, only when the "loss function" is hard to expressed mathematically and "error" is small when the two compared images are visual (not quantitative) similar, GAN is suitable.

In GAN matting, the predicted alpha mattes are the "counterfeit items", the ground truth alpha mattes will be the "real items". The generator of the GAN matting method works as the whole networks of previous matting methods based on deep learning, which aim at producing good enough matting results. The difference is: the GAN matting method uses a network (discriminator) to judge the quality of matting results, and back propagates the errors to the generator. However, this kind of judgement can be accomplished by using mathematical functions as loss functions, such as MSE, SAD, gradient and so on. Using a network to do the judgement and training both the generator and the discriminator is kind of waste. In addition, alpha matting is a pixel-wise accuracy work, we should exactly know the alpha value of each pixel in the unknown region. We should judge the matting results not only visually but also quantitatively.

In consideration of the inadequacy of GAN, we will solve the alpha matting problem via convolutional neural networks (CNNs), and the experimental results show that our method can get better matting results than GAN matting method.

Chapter 4

Alpha Matting via Residual Convolutional Grid Network

As we mentioned in the previous chapter, there are many drawbacks for conventional matting methods in extracting some special kinds of foreground objects. And the recently proposed deep learning based matting methods still have a large improvement space.

In this thesis, we propose a novel alpha matting method based on convolutional neural networks (CNNs). The proposed matting method is shown in Figure 4.1, which utilizes a convolutional grid network with residual learning framework [24] to realize an end-to-end matting. This method has achieved a state-of-the-art performance, which is next only to the best matting method: deep image matting method [65]. And the number of parameters in this method is less than one third of the number of parameters in the deep image matting method [65]. Our proposed matting method achieves good matting performance. We would like to try to add a refinement module on it, in order to solve the existed over smoothness problem and further improve our matting performance.

The experimental results of the proposed matting method, and the performance comparison with refinement module will be shown in Chapter 5. In this chapter, we will

describe the network architecture, calculate the trainable parameters, and present the loss function of the proposed method. We will also introduce the training and testing dataset, and describe the refinement module at the end.

4.1 The Proposed Matting Method

In this section, we will introduce the network architecture, trainable parameter calculation, and loss function of the proposed matting method: matting method based on the residual convolutional grid network.

4.1.1 The Overall Network Structure

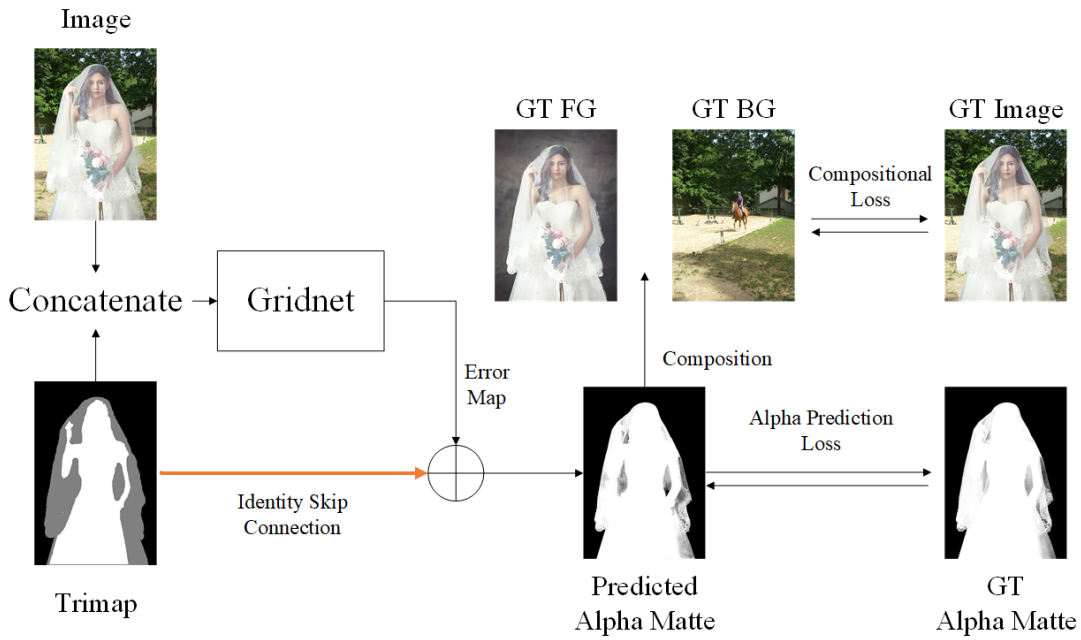


Figure 4.1: The overall network architecture. GT means the ground truth.

The overall network structure of our model is shown in Figure 4.1. This network structure is inspired by the residual learning framework [24], which predicts differences

between the input and the output, and is first proposed by Kaiming He (as Figure 4.2 shows).

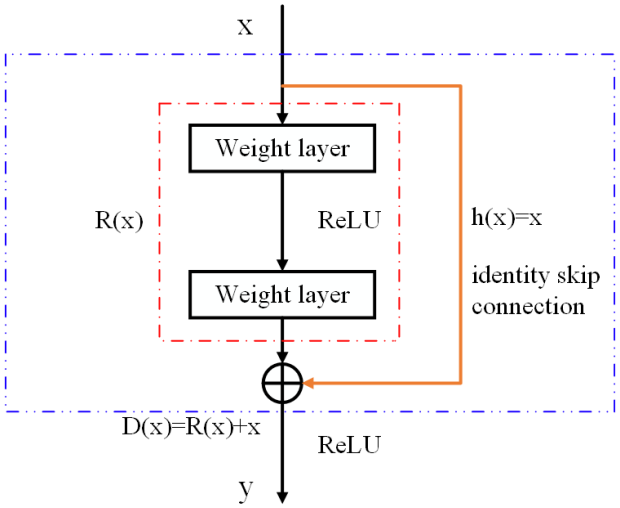


Figure 4.2: The residual learning framework [24].

As we know, with the development of CNNs, it has been discovered that increasing the depth of the network may increase the performance of the network in many data rich computer vision tasks. However, it has also been observed that along with the increase of network depth, the training becomes more and more difficult. Many problems appear and hamper further increase of the performance. One of the problem is the notorious vanishing/exploding gradients problem [3] [17] during the back-propagation steps. This problem may stop the network from further learning. The other problem is the degradation problem in training a deeper network. With the increase of network depth, accuracy gets saturated and then degrades rapidly. In other word, adding more layers to a network model leads to higher training and test errors. The degradation of accuracy is not due to overfitting.

To address these issues, residual learning framework is proposed. Figure 4.2 shows a residual learning framework [24]. The residual learning framework guides the stacked layers to learn residual function $R(x)$ with respect to the layer inputs x, instead of learning target

function $D(x)$ directly. The central idea of this design is as follows: when directly learning the desired function $D(x)$ is difficult, we can learn the residual function $R(x)$ instead. Then the desired function is obtained by $D(x) = R(x) + x$. The orange connection in Figure 4.2 is an identity skip connection ("shortcut"). In [25], various different types of mappings are used for the skip connections, and the identity mapping $h(x) = x$ is proved to be the best. With this skip connection, the gradient is back-propagated more easily. This residual learning architecture makes deep networks easier to optimize.

At the time the residual learning framework was first proposed, the residual learning network based on this framework had a striking performance in many computer vision tasks, such as ImageNet detection, ImageNet localization, MS COCO detection and segmentation. And now, the residual learning framework is still commonly used in all new proposed network architectures. In our network, we also use residual learning framework in many places, which we will introduce in detail in the following.

In our proposed matting method (as Figure 4.1 shows), we concatenate the three-channel image and the corresponding trimap along the channel to form the four-channel input and feed them to a convolutional grid network, which will be introduced in detail in the following subsection. The convolutional grid network extracts the low/mid/high-level semantic information progressively from the concatenation of the original image and the trimap, and synthesizes an error map. The error map represents the differences between the predicted alpha matte and the input trimap. The orange connection in Figure 4.1 is also an identity skip connection, which delivers all the pixel values of the trimap to the output of the convolutional grid network. There, the output of the convolutional grid network ("error map") has the same dimension with the trimap, and it will be pixel-wise added with the trimap to form the predicted alpha matte. This structure guides the Gridnet to focus on estimating the alpha values in the unknown regions, which effectively reduces the burden of Gridnet.

During the training, the original training images are cropped to 320×320 image patches,

so are the corresponding trimaps, before being fed to the overall network. The network is penalized by two loss functions: alpha prediction loss function and compositional loss function. During the testing, the images with original resolution and their same size trimaps will be fed to the network without cropping. The process can be expressed mathematically:

$$\mathrm{z} = \mathrm{x} + G\big(\mathrm{x}(+)\mathrm{y}\big), \tag{4.1}$$

where x is the trimap, y is the three-channel image, (+) represents concatenating along the channel, $G(\cdot)$ is the mapping in convolutional grid network and z is the predicted alpha matte.

4.1.2 Grid Network

The network that generates the error map is called convolutional grid network (Gridnet). In this subsection, we will introduce the inspiration and the structure of grid network.

The Inspiration of Gridnet

Our grid network is inspired by the network architecture proposed by Damien Fourure in [13]. The original Gridnet [13] works for image semantic segmentation, which aims at providing a class label for each pixel of an image, in order to segment an image into semantically meaningful regions. Semantic segmentation is a pixel-wise work, hence, the detail information of each pixel should be kept. Meanwhile, the high-level semantic information also needs to be extracted, in order to implement segmentation. To satisfy this kind of requirement, Damien Fourure proposed the grid network architecture in [13] for the first time. As its name shows, the grid network is organized as a two-dimensional grid. The "intersections" of grid are feature maps and the "edges" of grid are computation blocks. Every row of the grid is stack of blocks, which consist of convolutional networks. They keep the resolution of input and output of each row as constant, in order to keep the

detail information of each pixel. The columns in the left half of the grid are composed of downsampling blocks, which consist of classical convolutional networks with downsampling operations, which reduce the resolution of feature maps and enlarge the receptive field. The columns in the right half of the grid are composed of upsampling blocks, which consist of deconvolutional networks with upsampling operations, which increase the resolution of feature maps back to the ones before downsampling. By combining the rows and columns, the grid network retains the detail information of each pixel and extracts the high-level abstract information of the original image.

Our matting task has the same requirement as the semantic segmentation, we hope to keep the detail information of each pixel, in order to assign an alpha value for each pixel in the unknown region. Meanwhile, we also need to learn the high-level semantic information from the original image, in order to extract the foreground object accurately by recognizing the foreground item. Hence, we use the similar network architecture for our work and change the detailed compositions in each computation block of the original grid network so as to make the network adapt to our task well. The comprehensive experimental evidence provided in Chapter 5 indicates the rationality and validity of our network architecture in dealing with alpha matting problems.

The Structure of Gridnet

The grid network architecture of our model is shown in Figure 4.3, which is organized as a two-dimensional grid pattern, with m rows and n columns. The number of rows (m) and columns (n) in the network can be set by ourselves. We also call the rows as "streams". The intersections of the grid are stack of feature maps. We use $X_{i,j}$ to express the feature maps of line i and column j. Those feature maps are connected through computation blocks, and the information is processed in the computation blocks. Those computation blocks are placed horizontally or vertically with different colors.

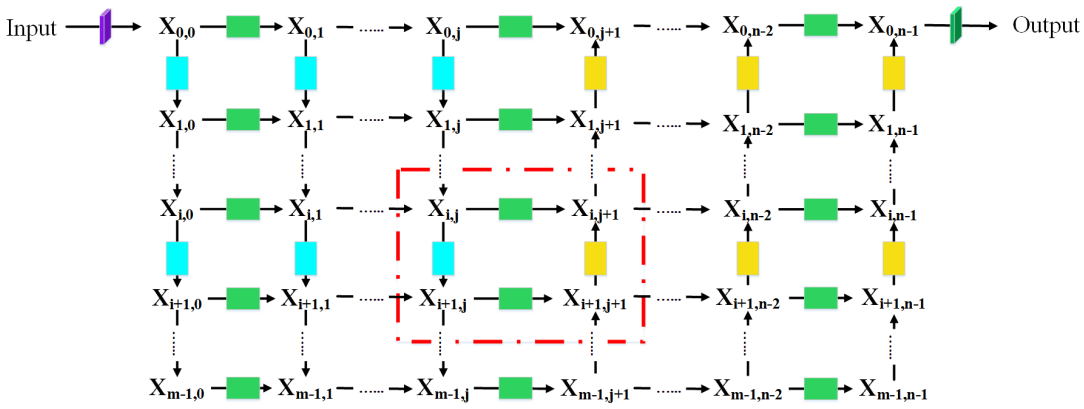


Figure 4.3: The grid network architecture.

The horizontal computation blocks (i.e. green blocks) are residual convolutional blocks, without upsampling or downsampling, they change neither the image resolution nor the number of feature maps. That means when the information pass through stream blocks, the resolution is a constant. The vertical computation blocks can be divided into two categories symmetrically. The left-half part are blue blocks and the right-half part are yellow blocks. Each blue block is a downsampling convolutional block, which doubles the number of feature maps and divides the size of feature map by two. On the contrary, each yellow block is an upsampling convolutional block, which doubles the spatial width and height of feature maps and halves the number of feature maps. The detailed configuration of those blocks will be described later.

During the training, the image patch and the corresponding trimap patch are concatenated along the channel dimension and to form the 4-channel input. The input passes through the first convolutional layer in stream 0 and output the feature map $X_{0,0}$. Then the information can be passed through any computation blocks along the arrows: If it chooses a straight way to only pass through the horizontal green computation blocks in stream 0, the resolution of feature maps will stay constant and the detail information of the original image can be reserved. If it passes through the vertical connections in the left-half part and goes back through the vertical connections in the right-half part, the downsampling

and upsampling operations reduce and recover the resolution of input image, respectively. Meanwhile, the downsampling operation increase the size of receptive fields significantly without enlarging the filter sizes, which will increase the number of parameters. Passing through the vertical connections means that the high-level abstract information can be better extracted and the more context information will be obtained from the original image.

By combining the different information flow paths (parallel paths and vertical paths), we hope the network can either retain the detail of the image or learn the high-level semantic information and context information of image. And we also hope that the network itself can decide the paths to pass the information, in accordance with its requirements. After choosing different ways, the information is finally converged as $X_{0,n-1}$, these feature maps are then passed through the convolutional layer on the "tail" of the stream 0, and output a 1-channel error map. The error map adds with the trimap to form the final predicted alpha matte.

During the training, assuming that the information is passed along the blue paths show in Figure 4.4, then the gradient will be back propagated along the red paths in Figure 4.5, and the parameters in the convolutional layers along the red arrows will be updated.

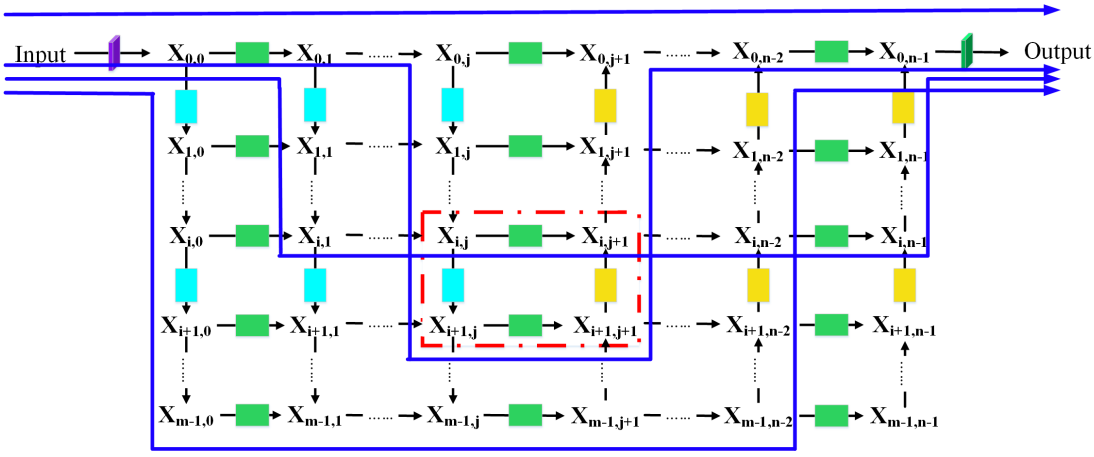


Figure 4.4: A forward propagation example of grid network.

To further explain our network architecture and the rule of passing the information,

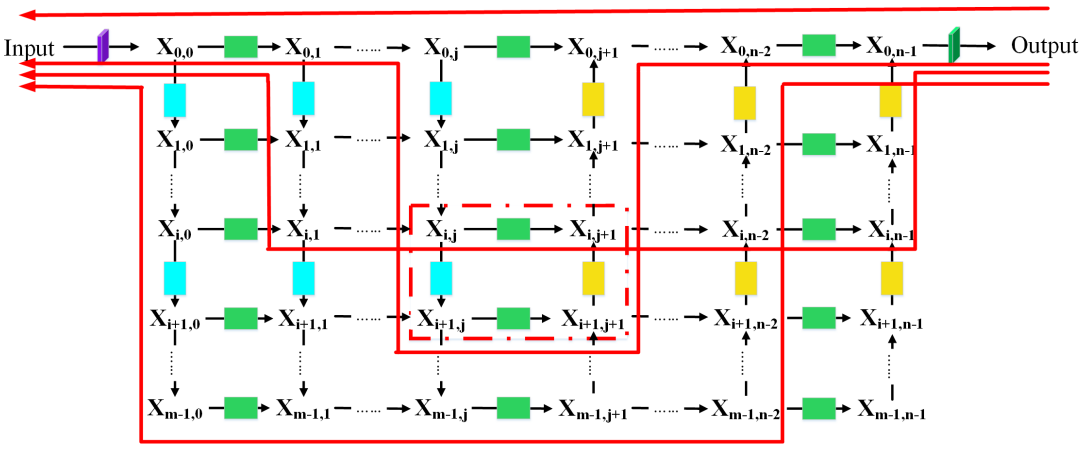


Figure 4.5: The corresponding backward propagation example of grid network.

we magnify the red square in Figure 4.3 and show the detailed configuration of network in Figure 4.6.

Each green block is a residual convolutional block [24], where we uses two-layer 3×3 convolutional natural networks with stride one and one-pixel padding, to calculate the differences between the outputs and the inputs of the block. ReLU is the non-linear layer, and is set before each convolutional layer. We set the kernel size of the filter to 3×3 to minimize the number of parameters, for 3×3 is the smallest size to capture the notion of up/down, left/right and center. The 3×3 filter is convolved with the input at every pixel (with stride one) and the consecutive stack of two 3×3 convolutional layers have achieved an effective receptive field of 5×5. Since the convolution stride is fixed to one pixel in both of the two convolutional layers, and the spatial padding of convolutional layer input is one pixel, (which is realized by reflecting one-pixel value at the margin of image patch), the spatial resolution of feature maps are constant between inputs and outputs.

Mathematically, we use $G(\cdot)$ to express the convolution operation for the residual block, θ^G are some trainable parameters in residual block. When we fed the block with a feature tensor $X_{i,j}$ of dimension $(F_{i,j} \times W_{i,j} \times H_{i,j})$, where $F_{i,j}$ is the number of feature maps and $W_{i,j}$ is the width of the map and $H_{i,j}$ is the height of the map. Then the output of the residual block is $X_{i,j} + G(X_{i,j}, \theta^G_{i,j})$, which is a tensor of the same dimension as $X_{i,j}$.

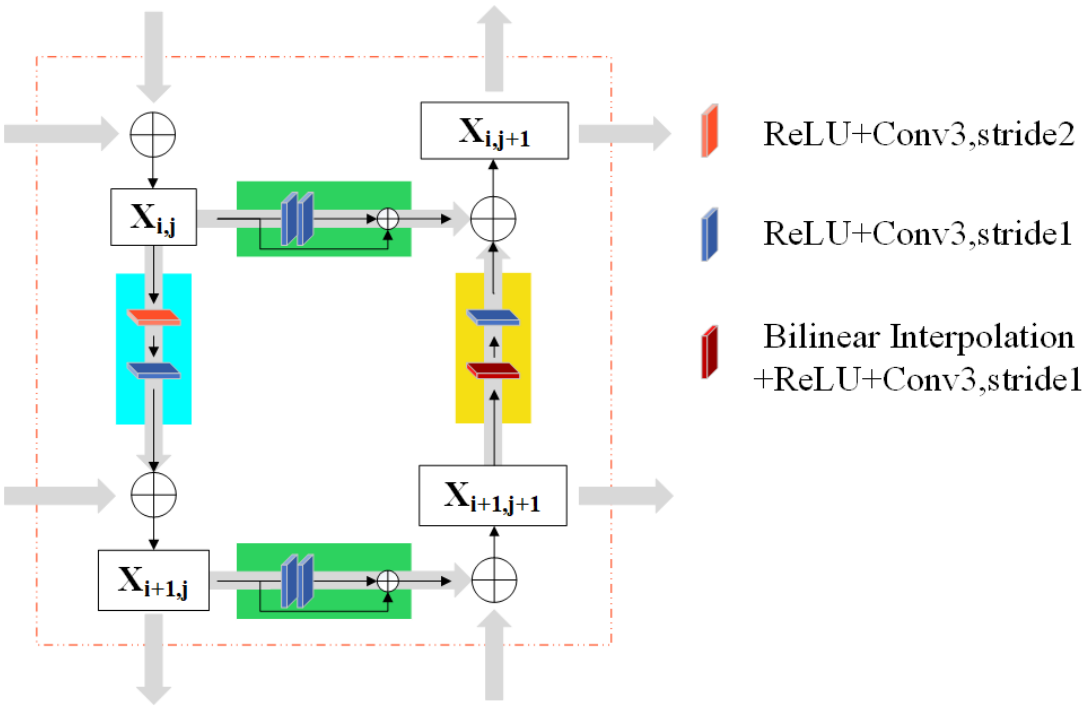


Figure 4.6: Detail configuration of grid network.

All the vertical blocks are not residual anymore. Each vertical blue block is a downsampling convolutional block, which also contains two convolutional layers with a kernel size of 3×3. The spatial padding of convolutional layers input is also one pixel. We use reflection to realize the padding. ReLU is the non-linear layer, and is set before each convolutional layer. The first convolutional layer has stride two, as a result, the spatial width and height of feature maps are reduced by two when they are passed through the first convolutional layer. Meanwhile, the number of filters in the first convolutional layer is the twice of the number of input feature maps, so the number of the output feature maps are doubled by the first layer at the same time. The second convolutional layer in the vertical blue block has stride one, and the same number of filters as the first convolutional layer, so the volume of feature maps is unchanged when they pass through the second convolutional layer. We use $B(\cdot)$ to represent the mapping operation of the downsampling convolutional block. Assume that the input of the block is the feature map $X_{i,j}$ of dimension $(F_{i,j} \times W_{i,j} \times H_{i,j})$, θ^B are some trainable parameters in the blue block, then the output is $B(X_{i,j}, \theta^B_{i,j})$, which

is a tensor of dimension $(2F_{i,j} \times W_{i,j}/2 \times H_{i,j}/2)$.

On the contrary, each yellow block is an upsampling convolutional block, it first uses bilinear interpolation method to increase the size of the feature maps by a factor 2. After the size of the feature maps recovers to the one before downsampling operation, the feature maps go through two successive convolutional layers, also with a kernel size of 3×3 and stride one. The spatial padding of convolutional layers input is also one pixel and realized by reflection. ReLU is the non-linear layer and is set before each convolutional layer. The number of filters in these two layers are the half of the number of input feature maps. As a results, these convolutional layers decrease the number of feature maps by two. Hence, the upsampling computation blocks augment the feature map's width and height by two, and reduce the number of feature maps by 2, which makes the dimension of tensor back to the original one (i.e. the one in the upper stream), to allow the addition. The function $Y(\cdot)$ is used to defined the operation of the yellow block, θ^Y are the trainable parameters in the yellow block, after passing data $X_{i+1,j+1}$ through the yellow block, the output $Y(X_{i+1,j+1}, \theta^Y_{i+1,j+1})$ is of dimension $(F_{i+1,j+1}/2 \times 2W_{i+1,j+1} \times 2H_{i+1,j+1})$.

Except for the border feature maps, each feature map $X_{i,j}$ in the grid is the sum of two different kinds of computations: one is the horizontal residual computation processing the information from its left neighboring feature map $X_{i,j-1}$, and the other is the vertical downsampling (upsampling) convolutional computation processing the information from its upper (lower) neighboring feature map $X_{i-1,j}$ ($X_{i+1,j}$). The second computation depends on if the column is a downsampling or upsampling one.

If $X_{i,j}$ is in a downsampling column:

$$X_{i,j} = X_{i,j-1} + G(X_{i,j-1}, \theta^G_{i,j-1}) + B(X_{i-1,j}, \theta^B_{i-1,j}). \tag{4.2}$$

Otherwise, if $X_{i,j}$ is in an upsampling column:

$$X_{i,j} = X_{i,j-1} + G(X_{i,j-1}, \theta^G_{i,j-1}) + Y(X_{i+1,j}, \theta^Y_{i+1,j}). \tag{4.3}$$

The border feature maps are produced more simply than the interior ones in the grid. Here, we use a specific Gridnet example to illustrate them. In our experiment, we adjust the number of rows, columns and the feature maps in the first stream of the grid network, in order to obtain the best matting result with an acceptable number of parameters. We find that the grid network with five rows (5r), four columns (4c) and 16 feature maps (16f) in the first stream works best. So we show that grid network in the Figure 4.7. And show

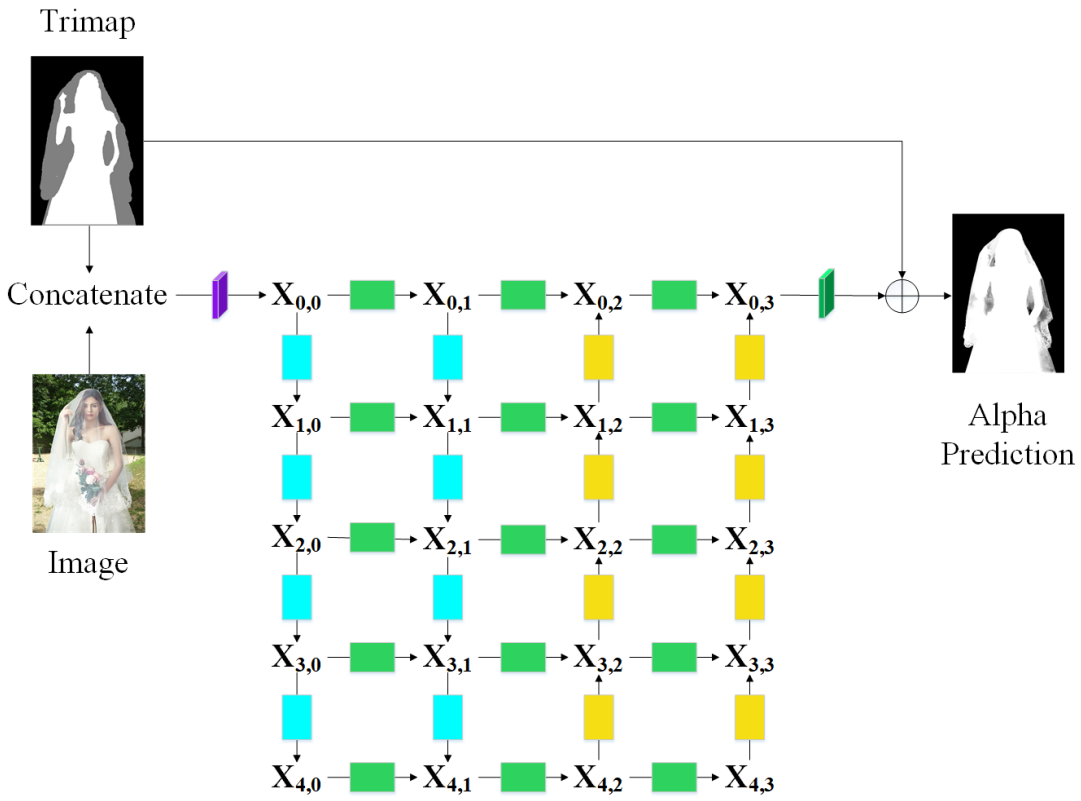


Figure 4.7: Grid network architecture with 5r4c16f.

the first column of this network in Figure 4.8 and the last column of this network in Figure 4.9. The configuration of network in the second column is totally the same as the first one and the third column is the same as the last one.

For the first row, during the training, the input to our grid net is a fixed-size of 320×320, 4-channel image patch, which is formed by concatenating the original 320×320, 3-channel RGB image patch with its corresponding trimap along the channel dimension. Hence, the

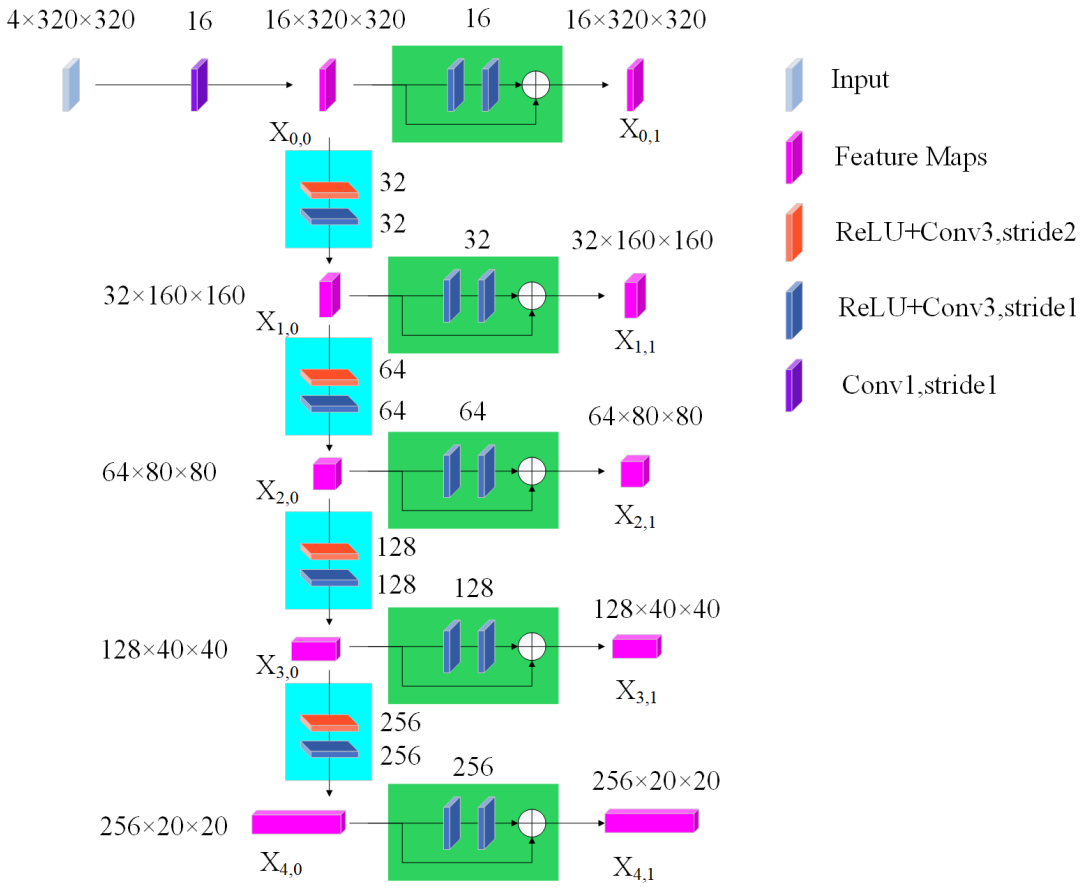


Figure 4.8: The first column of grid network with 5r4c16f.

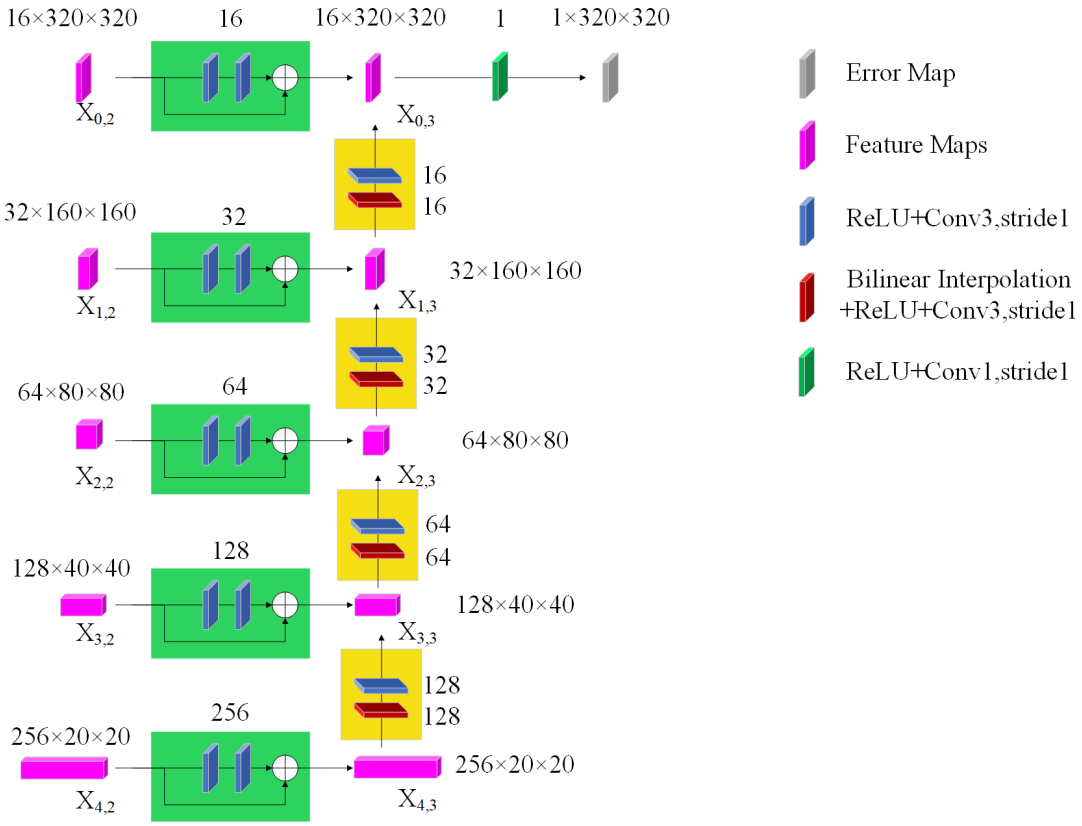


Figure 4.9: The last column of grid network with 5r4c16f.

input volume is $4 \times 320 \times 320$. The concatenated input is first fed to a convolutional layer, which has 16 different filters, with a kernel size of 1×1 and stride one. So the output feature maps of this first convolutional layer: $X_{0,0}$ has the volume of $16 \times 320 \times 320$. The first convolutional layer aims at adjusting the dimension of the input in order to be processed conveniently in the grid network. Then, we set each convolutional layer in the green residual block in the first stream to have 16 different 3×3 filters, with stride one and one-pixel padding. ReLU is the non-linear layer and is set before each convolutional layer. As a consequence, the volume of all the feature maps in the first stream is $16 \times 320 \times 320$. Since the finally output is a 1-channel error map, the last feature maps of the first stream $X_{0,3}$ is arranged to go through a convolutional layer with just one filter, with a kernel size of 1×1 and stride one. ReLU is the non-linear layer and is set before the convolutional layer. The 1-channel error map will be pixel-wise added with trimap to produce the predicted alpha matte. We should notice that, due to the residual block does not change the volume of feature maps, feature maps in the same stream have the same volume.

For the first column, to double the number of feature maps when they are passed through the vertical blue block, we set the number of filters of convolutional layers in the first blue block to be 32, 64 for the second block, 128 and 256 for the third and the fourth blue block, respectively. The feature map passes through these four vertical blue block and ends with a volume of $256 \times 20 \times 20$. Figure 4.8 shows the volume of feature maps and the number of filters in each convolutional layers of each block in the first column.

Meanwhile, for the last column, the yellow blocks upsample the feature maps back to the volume before downsampling. So we set the number of filters in convolutional layers of each block in the fourth column to be 128, 64, 32 and 16, respectively, from the bottom one to the top one. Figure 4.9 shows the volume of feature maps and the number of filters in each convolutional layer of each blocks in the last column.

As we mentioned above, we can choose the number of rows, columns, and feature maps of the first row by ourselves. We should notice that the number of downsampling convolu-

tional column and the number of upsampling convolutional column should be consistent, which means that the number of columns should be an even number. After setting the number of feature maps in the first row, the number of feature maps in other rows are fixed: twice the number of the upper row. We have changed those adjustable parameters in our experiment, and the corresponding experimental results are shown in the Chapter 5.

4.1.3 Trainable Parameter Calculation

In neural networks, the number of trainable parameters influences the computation complexity and memory usage. Our Gridnet structure controls this number well by adjusting the number of the rows/streams, the downsampling columns, the upsampling columns, and the feature maps in the first row. In this subsection, we provide the counting method of trainable parameters, and calculate the number of trainable parameters of $5r4c16f$ grid network.

Assuming our grid network has M rows/streams, N_d downsampling columns, N_u upsampling columns, and F_0 is the number of feature maps at resolution $W_0 \times H_0$ in the first row/stream. The number of channel of input is C_i, the number of channel of output is C_o. The kernel size of convolutional layer in the "head" and "tail" are k_{head} and k_{tail}, respectively. The kernel size of convolutional layer in the grid is k. The input has a volume of $4 \times W_0 \times H_0$ and the output has a volume of $1 \times W_0 \times H_0$. The following formulas can be used to calculate trainable parameters of different grid networks with different number of rows, columns, feature maps and so on. For our proposed $5r4c16f$ grid network, $M = 5$, $N_d = 2$, $N_u = 2$, $F_0 = 16$, $C_i = 4$, $C_o = 1$, $k_{head} = 1$, $k_{tail} = 1$ and $k = 3$.

For the first convolutional layer (head) and the last convolutional layer (tail) in the first row/stream, the number of trainable parameters are given by:

$$p_{head} = C_i \times k_{head} \times k_{head} \times F_0 = 4 \times 1 \times 1 \times 16 = 64, \tag{4.4}$$

$$p_{tail} = F_0 \times k_{tail} \times k_{tail} \times C_o = 16 \times 1 \times 1 \times 1 = 16. \tag{4.5}$$

For all the green block in all the M rows, the number of trainable parameters p_r are given by :

$$\begin{aligned} p_r &= 2\{(2^0)^2 + (2^1)^2 + (2^2)^2 + (2^3)^2 + ... + (2^{M-1})^2\}(F_0 \times k \times k \times F_0)(N_d + N_u - 1) \\ &= \frac{2}{3}(4^M - 1)(F_0 \times k \times k \times F_0)(N_d + N_u - 1) \\ &= \frac{2}{3}(4^5 - 1)(16 \times 3 \times 3 \times 16)(2 + 2 - 1) \\ &= 4,713,984. \end{aligned} \tag{4.6}$$

For all the blue blocks in all the N_d downsampling columns, the number of trainable parameters p_d are given by:

$$\begin{aligned} p_d &= \{2^1 + 2^2 + 2^3 + ... + 2^{2(M-1)}\}(F_0 \times k \times k \times F_0)(N_d) \\ &= 2(2^{2(M-1)} - 1)(F_0 \times k \times k \times F_0)(N_d) \\ &= 2(2^{2(5-1)} - 1)(16 \times 3 \times 3 \times 16)(2) \\ &= 2,350,080. \end{aligned} \tag{4.7}$$

For all the yellow blocks in all the N_u upsampling columns, the number of trainable parameters p_u are given by:

$$\begin{aligned} p_u &= \{2^0 + 2^1 + 2^2 + ... + 2^{2(M-1)-1}\}(F_0 \times k \times k \times F_0)(N_u) \\ &= (2^{2(M-1)} - 1)(F_0 \times k \times k \times F_0)(N_u) \\ &= (2^{2(5-1)} - 1)(16 \times 3 \times 3 \times 16)(2) \\ &= 1,175,040. \end{aligned} \tag{4.8}$$

The total parameters in our grid network with 5 rows, 4 columns, 16 feature maps in the first row is $8,239,184$, which is less than one-third of the number of parameters in deep image matting method without refinement [65], which is more than 26 millions.

4.1.4 Loss Function

Our proposed matting method, using a convolutional grid network to predict alpha matte directly from image and trimap, leverages two losses to form the overall loss function, which is inspired by [65].

The first loss is the alpha prediction loss, which is improved from L1 loss function. The L1 loss function measures the sum of the absolute difference between each pixel value in the ground truth alpha matte and the corresponding pixel value in the predicted alpha matte. The formula is given by:

$$\mathrm{L1}(\alpha_{pre}, \alpha_{gt}) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \Big| \alpha_{pre}(i,j) - \alpha_{gt}(i,j) \Big|, \tag{4.9}$$

where $|\cdot|$ denotes the absolute value, $\alpha_{gt}(i,j)$ denotes the pixel value of the ith row, jth column in ground truth alpha matte, $\alpha_{pre}(i,j)$ denotes the pixel value of the ith row, jth column in predicted alpha matte. The size of ground truth alpha matte is $M \times N$.

However, because the absolute values are non-differentiable, we improve the L1 loss function to a differentiable one, called alpha prediction loss:

$$\mathrm{L1}^{alpha}(\alpha_{pre}, \alpha_{gt}) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sqrt{(\alpha_{pre}(i,j) - \alpha_{gt}(i,j))^2 + \sigma^2}, \tag{4.10}$$

where $\alpha_{pre}(i,j), \alpha_{gt}(i,j) \in [0,1]$, σ is a small value set to be 10^{-6} in our experiments. Then the derivative of $\mathrm{L1}^{alpha}$ with respect to $\alpha_{pre}(i,j)$ is:

$$\frac{\partial \mathrm{L1}^{alpha}(\alpha_{pre}, \alpha_{gt})}{\partial \alpha_{pre}(i,j)} = \frac{\alpha_{pre}(i,j) - \alpha_{gt}(i,j)}{\sqrt{(\alpha_{pre}(i,j) - \alpha_{gt}(i,j))^2 + \sigma^2}}. \tag{4.11}$$

The second loss is compositional loss, which is also improved from the L1 loss function. At this time, the L1 loss function calculates the sum of the absolute difference between each pixel value in the ground truth RGB color image and the corresponding pixel value in the compositional predicted RGB color image:

$$\mathrm{L1}(I_{pre}, I_{gt}) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \Big| I_{pre}(i,j) - I_{gt}(i,j) \Big|, \tag{4.12}$$

where, I_{gt} is the pixel value of the RGB 3-channel ground truth image, which is formed by compositing ground truth foreground object, the ground truth alpha matte with the ground truth background image. And I_{pre} is the pixel value of the RGB 3-channel predicted image, which is formed by compositing the ground truth foreground image, the predicted alpha matte with the ground truth background image. To make the loss function differentiable, we use the compositional loss function to approximate the L1 loss function:

$$\mathrm{L1}^{comp}(I_{pre}, I_{gt}) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\sqrt{(I_{pre}(i,j) - I_{gt}(i,j))^2 + \sigma^2}, \tag{4.13}$$

where σ is also set to be 10^{-6} in our experiments. The compositional loss forces the network following the matting function in Equ. (1.1), in order to obtain more accurate alpha matte.

The overall loss is the weighted sum of these two loss functions, in our experiment, we assign the same weight to these two loss functions:

$$\mathrm{L}_{overall} = 0.5\mathrm{L1}^{alpha}(\alpha_{pre}, \alpha_{gt}) + 0.5\mathrm{L1}^{comp}(I_{pre}, I_{gt}). \tag{4.14}$$

The reason why we rely on the L1 loss function but not the L2 loss function is that the L2 loss function performance bad if the outliers existed. The L2 loss function takes the square of the differences, which inevitably lead to the outliers having larger influence on the L2 loss function than the L1 loss function. Hence, the network with the L2 loss function may try to minimize the error caused by outliers, at the expense of increasing the error of common examples. In our task, outliers can be tolerated, hence, the L1 loss function is adopted, which is more robust to outliers.

4.2 Dateset

Our dataset contains two parts: training dataset and testing dataset. Before the deep-learning based matting methods, the conventional matting methods evaluate their matting performance by using the dataset provided by the alpha matting evaluation website

alphamatting.com [47]. This dataset has achieved tremendously success in accelerating the pace of development in alpha matting research. However, there are two defects in the dataset provided by the website:

- The number of dataset is two small, it is not enough to train a neural network. It contains only 27 images for training and 8 images for testing. If the network is a little bit complex, the small training dataset will lead to overfitting. This is the reason why the DCNN matting method [7], which uses this dataset as training and testing dataset, has a simple CNN architecture: a stack of 6 convolutional layers.

- The content of the image of dataset is severely limited in its diversity and restricted to lab scenes. Figure 4.10 and Figure 4.11 shows some training images and testing images, respectively. We can find that no one is a real natural image. They have two common characteristics: 1) The foreground objects are static and single: dolls, stuffed toys, artificial potted plant, pen holder, net, plastic bag and so on. There are no human beings or animals. In addition, those foreground objects rarely appear in common natural image. 2) The dataset are created in indoor static environment. The backgrounds of images are artificial. They are pictures with natural scenery, or indoor scene. The backgrounds are also blurry, which may bias the network to recognize the blur of images and choose the clear objects as foreground and blurry parts as background. As a result, performing well in this dataset may not be equal to having good results in dealing with a real natural image.

In general, the dataset for natural image (alpha) matting based on deep learning should have several important properties. Firstly, the number of training dataset should be large enough to allow for the training of network. Secondly, the images in dataset should be like natural images, with natural lighting, natural background and foreground objects common seen in real world. Thirdly, the foreground objects should be diversified, with human beings, animals, plants, industrial products and so on. Fourthly, a variety of

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

(e) (f) (g) (h)

Figure 4.10: Training images provided by matting evaluation website *alphamatting.com* [47]. In the purpose of keeping the layout neat, the resolution of images has been adjusted.

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

(e) (f) (g) (h)

Figure 4.11: Testing images provided by matting evaluation website *alphamatting.com* [47]. In the purpose of keeping the layout neat, the resolution of images has been adjusted.

difficult conditions found in real world image should be covered, such as color distributions overlapping, complicated background, transparent and reflective phenomena. Fifthly, the span of the resolution of images should be large, in order to deal with the natural images with high resolution. Finally, the dataset should be paired with high quality ground truth alpha matte, to allow for training and fair comparison.

In consideration of the flaws in the existed matting dataset, a bigger and more suitable dataset for alpha matting methods based on deep learning are needed. However, there is no standard method for producing the ground-truth alpha mattes for images. Because producing the matting ground truth has strict restrictions, which is significantly much more difficult than for most other computer vision tasks (e.g. labeling the objects of millions images manually in order to realize image classification or image recognition and so on). In addition, the quality of the ground-truth alpha matte needs to be very high, it should define every pixel's alpha value, which is a pixel-wise accuracy.

At the very beginning, we tried to produce the ground truth by ourselves, through chroma keying technique. We set the foreground objects in front of the solid color background (green screen in our lab) with the constraint that there are no similar colors between background and the foreground objects. Theoretically, with the simple background color and the easily separable foreground and background color distribution, the chroma keying technique can accurately and efficiently extract the foreground objects and obtain the alpha mattes by subtracting the background color in each pixel.

However, the results are not satisfactory. The reason has three aspects: 1) the background can not be completely monochromatic and evenly lighted. 2) The defects in current algorithms limit the accuracy. 3) This kind of work is really time consuming. In professional studio, the background can be constant color by placing the lighting source evenly. On the contrary, in practice, we inevitably have shadow and crumple on the background surface and the lighting condition is also uneven. In addition, the current algorithms or keying apps perform badly when they encounter the situation that the background light partially

passes through the foreground, reflects on the foreground, or the foreground objects have similar color with the background (green screen). Meanwhile, producing a ground truth image need user interaction, it may take hours to refine the matting result in order to get an accurate ground truth. However, we do not have enough time to produce hundreds of ground truth alpha mattes and the limitation of accuracy in producing the ground truth alpha mattes also restricts the accuracy of our results, because we use these ground truth alpha mattes as "answers" to train our network.

Due to the strict restriction of generation of ground truth images and the time limitation of our research, in our method, we use the training and testing dataset provided by the deep image matting method [65]. This method [65] provides us with 431 foreground objects and the corresponding ground truth alpha mattes for training, and 1000 composition images, trimaps and ground truth alpha matte for testing. The 431 training foreground objects contain the 27 foreground objects provided by [47], every fifth frame from the videos of [51] and some other objects extracted from real images with simple or plain backgrounds. These pure foreground colors and the corresponding alpha mattes are generated by Photoshop. These foreground objects contain rich content and multiple diversity. As shown in Figure 4.13, they cover more matting cases, such as transparency, semi-transparency, perforated structure, hair, fur and so on. Then we composite each foreground object onto 100 background images, based on ground truth alpha matte and matting function, to form a training dataset, which contains $431 \times 100 = 43100$ images. Those background images are selected randomly but not repeatedly from MS COCO [36] 2014 released training dataset. The MS COCO dataset are proposed for segmenting individual object instances, and comprised of images depicting complex everyday scenes of common objects in their natural context. The MS COCO images dataset contains multiple object categories, each object category contains a number of instances. And a majority of images in this dataset are non-iconic images, which means that a lot of objects are in an unconventional context, with partially occluded, mirror, reflection, or clutter in back-

ground. Figure 4.12 shows some images in the MS COCO 2014 released training dataset.

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

(e) (f) (g) (h)

Figure 4.12: Some example-images of MS COCO dataset [36].

Since the background images have multiple objects and complex context, when they are composited with the 431 foreground objects, the composited images are more complicated. They may have similar foreground and background colors and complex background textures, which are also challenging for conventional matting algorithms. Figure 4.13 shows some representative examples of our training dataset and the corresponding alpha mattes. The resolution of training images covers a wide range, from hundreds to $4k$, which is also much bigger than the training dataset provided by the alpha matting evaluation website *alphamatting.com* [47]. For viewing convenience, the images shown in Figure 4.13 have been reduced in resolution.

For fair comparison, we use the testing dataset that is the same as the one deep image matting method [65] used. This testing dataset is called the Composition-1k testing dataset, which contains 1000 images, generated by compositing each of 50 foreground objects onto randomly but not repeatedly sampled 20 background images from Pascal VOC [12] image dataset, based on ground truth alpha matte and matting function. The

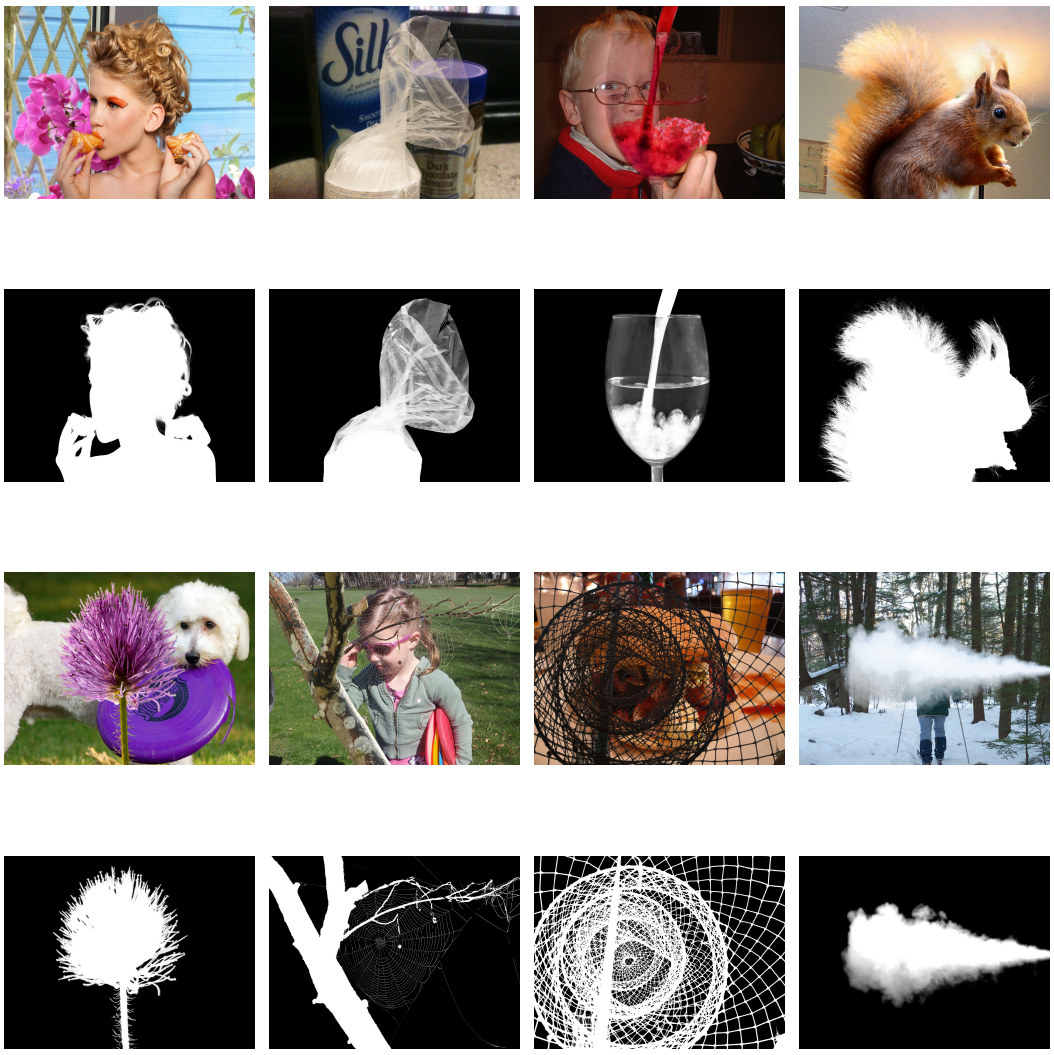


Figure 4.13: Some example-images and the corresponding alpha mattes of our training dataset. The resolution of each image has been reduced for easily displaying.

deep image matting method also provided us with trimaps for testing dataset. There are 20 trimaps for each testing foreground image, one for each of the backgrounds on which the foreground was composited.

We should notice that, when compositing the foreground objects on the background images, we set the width and height of each foreground as w and h, respectively, and the width and height of the background image as bw and bh. Then we calculate the ratio $R = max[\frac{w}{bw}, \frac{h}{bh}]$, if $R > 1$, we resize the background image to be the same as or bigger than the size of the foreground image by Bicubic interpolation method.

4.3 Refinement

In this section, we will introduce the refinement module, including the overall network architecture after adding the refinement module and the architecture of the refinement network in the refinement module.

4.3.1 The Overall Network Structure

Although the grid network proposed in the previous section has produced the state-of-the-art alpha matting results, we observed that over smoothness still existed in the margin of some foreground objects. We hope to improve our matting results by adding a refinement module in our network, to solve the over smooth problem. Hence, the overall network architecture of the matting method with refinement consists of two modules: Gridnet module and a refinement module, as shown in Figure 4.14.

The previously proposed network architecture is used as the Gridnet module, which extracts features from input and produces an initially predicted alpha matte. This initially predicted alpha matte will be enhanced by the refinement module to obtain the final result.

The refinement module also utilizes the residual learning framework [24]. The network that generates the error map in the refinement module is call refinement network. This refinement network extracts the detail information, missed by the Gridnet module, from the original image and initially predicted alpha matte. To explain it more clearly, in the process of refinement, the initially predicted alpha matte is concatenated with the original 3-channel image, and then fed to the refinement network. The output is a 1-channel error map, which can be seen as supplement or fine tuning of the initially predicted alpha matte. So it is added with the initially predicted alpha matte to produce the refined matting results.

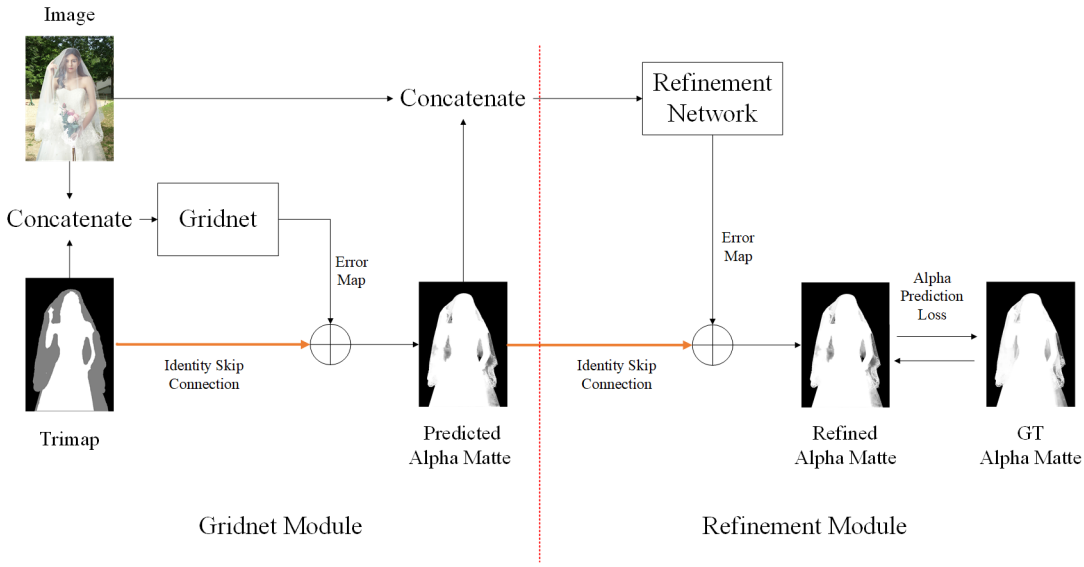


Figure 4.14: The overall network architecture. GT means the ground truth.

4.3.2 Refinement Network Architecture

We have tried two different refinement networks, the first is inspired by the refinement network proposed by [65], and the second is proposed by ourselves.

Four Convolutional Layers. The first refinement network architecture is shown in Figure 4.15. It is the same as the one the deep image matting method [65] used. Since this refinement network structure sharpens the edges effectively in their method, shows a better performance than Guided filter [22] and improves the alpha matting results, we use it in our method hoping to solve the over smooth problem in the marge of foreground objects.

The refinement network contains four successive convolutional layers, as Figure 4.15 shows. These four convolutional layers with the same kernel size: 3×3. The former three convolutional layers have 64 filters and followed by ReLU layers, and the last layers only has 1 filter and no non-linear layer followed. The deep image matting method uses these simple layers to supplement the detail information missed by their alpha prediction stage, and to sharpen the edges of alpha matte. However, our experimental results shown in Chapter 5 indicate that this network structure is useless in our method. It not only fails to sharpen the edges but also confuses our network and reduces the accuracy of alpha matte.

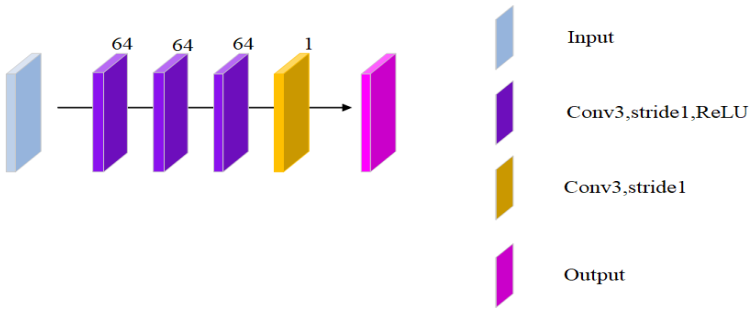


Figure 4.15: The four convolutional layers for refinement.

Grid Network with Two Rows and Four Columns. Considering that the first refinement network structure is not useful, we propose a new refinement network structure hoping to solve the over-smooth problems and improve our matting results. The second refinement network architecture is shown in Figure 4.16. The Gridnet has been proved to have the ability to learn the semantic information of image. Hence, we use a simple

Gridnet as the refinement network. We hope that this simple Gridnet can learn the edge information of the image specifically, in order to sharpen our matting results.

The refinement network is a two rows and four columns ($2r4c$) Gridnet. The 3-channel original image and the 1-channel predicted alpha matte are concatenated to form a 4-channel input, which is passed through a convolutional layer with a kernel size of 1×1 and stride one. Then the output $X_{0,0}$ is fed to a grid network with $2r4c$, and the number of filters in the convolutional layers in the first row are 16. After that, the output of the grid network $X_{0,3}$ is passed through a convolutional layer, which has one filter, with a kernel size of 1×1 and stride one. ReLU is the non-linear layer, and is set before the last convolutional layer. Finally, a 1-channel error map is output from the refinement network, which will be added with the initially predicted alpha matte to form the refined matting result. The input of interior convolutional layers in the Gridnet are padding with one pixel around the edge. The padding is realized by reflection.

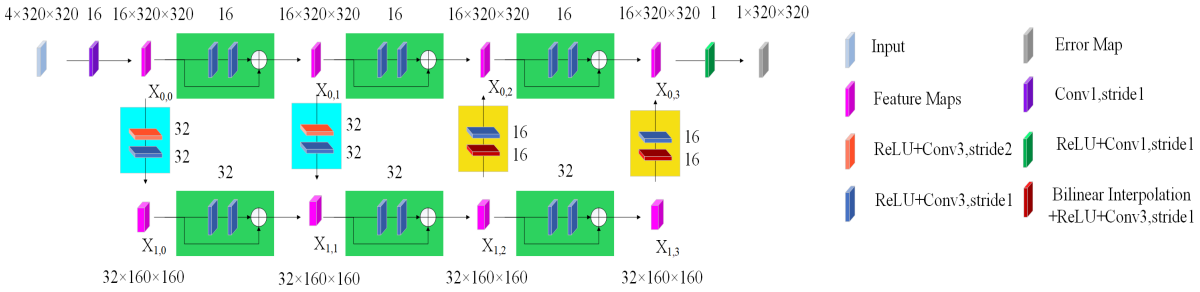


Figure 4.16: The $2r4c$ grid network for refinement.

Chapter 5

Experimental Results

In this chapter, we will introduce our experiment setup and implemention, evaluation criteria, then show and analyze the experimental results.

5.1 Experiment setup

The experiment platform is a PC equipped with an Intel Core i7-7700K CPU @ 3.60 GHz×8, 16 GB system memory, 512 GB Solid State Disk, and a single NVIDIA GeForce GTX 1080Ti graphical adapter with 11 GB memory. The operating system is Debian GNU/Linux version 9.4. The software framework is PyTorch [42] version 0.3, which is an open-source library designed to enable rapid research on machine learning models. PyTorch provides Tensor computation with strong GPU acceleration. To realize the GPU acceleration computation, NVIDIA proprietary driver and GUDA toolkit are installed. The language is Python 3.6 installed by Anaconda.

5.2 Experimental Implemention

The training dataset has only 431 unique foreground images, although each foreground image has been composited with 100 different background images (based on ground truth alpha matte and matting function Equ. (1.1)), the 43100 images is also far less than the training dataset for other computer vision tasks, such as ImageNet. Hence, some training strategies must be used to improve the variation of training dataset, in order to avoid overfitting and make full use of these precious training dataset:

- We get trimap by using a Gaussian filter on ground truth alpha matte with random radius from 5 to 30. We set the pixel values in the unknown region (i.e. where the pixels with value $\in (0, 255)$) to be 128. Producing trimaps by dilating ground truth alpha matte randomly can save our time and make our network to be more robust.

- We randomly crop $n \times n$ (foreground, provided ground truth alpha matte, background, as well as trimap) patches pairs centered on unknown regions. This focus our learning on key regions. To achieve this, we randomly select a pixel in the unknown region of trimap as the central point of the patches, and spread a square centered at that point. The side length of the square is set previously. In this process, if the square touches the edges, it will stretch the rest length on the opposite direction.

- We change the cropping size of the square patches pairs from 320×320 to 480×480 randomly and resize them to 320×320. This improves our network's robustness to scales, and makes our network to learn the semantic and context information better.

- Before we compositing the foreground images with background images (based on the ground truth alpha matte and matting function Equ. (1.1)), we randomly horizontally flip the foreground and the ground truth alpha matte, or background with fifty percent chance, which leads to four different combinations with the same foreground and background patches pairs.

- For each epoch, we shuffle the dataset, randomly pick up image and trimap pairs in 43,100 training dataset and crop them in random place with random size.

These training dataset augmented strategies increase the variation and amount of training dataset effectively.

5.2.1 Matting Method without Refinement

When training the Gridnet without refinement, we initialize the whole network with Xavier random variables and pad one pixel on each input of interior convolutional layers in the grid, by reflecting one pixel at the margin of the image (or feature map). We concatenate the 3-channel 320×320 image patch and corresponding trimap patch along the channel to form the 4-channel 320×320 input. We feed the Gridnet with the input and add the output of the gridnet with trimap patch to form the predicted alpha matte. All the training strategies proposed previously are used. The network is penalized by the overall loss function $L_{overall}$ defined by Equ. (5.1).

$$\begin{aligned} \mathrm{L1}^{alpha}(\alpha_{pre}, \alpha_{gt}) &= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sqrt{(\alpha_{pre}(i,j) - \alpha_{gt}(i,j))^2 + \sigma^2}, \\ \mathrm{L1}^{comp}(I_{pre}, I_{gt}) &= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sqrt{(I_{pre}(i,j) - I_{gt}(i,j))^2 + \sigma^2}, \\ \mathrm{L}_{overall} &= 0.5\mathrm{L1}^{alpha}(\alpha_{pre}, \alpha_{gt}) + 0.5\mathrm{L1}^{comp}(I_{pre}, I_{gt}). \end{aligned} \tag{5.1}$$

In the experiments, Adam [30] optimizer is used with learning rate 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$. The mini-batch size is 8. The program takes 0.064 second to process one training image patch. We trained about 200 epoches before the Gridnet is converged. After convergence, we finish training and fix its parameters.

During testing, we also concatenate the original image and its corresponding trimap as 4-channel input for testing. However, we reserve the resolution of the original image and

do not crop them anymore. Then a forward-propagation of the network is performed to output the alpha matte prediction.

5.2.2 Matting Method with Refinement

As we described in Section 4.3, the network architecture of matting method with refinement consists of two modules: Gridnet module and a refinement module (as shown in Figure 4.14). To explicitly guide the behavior of the refinement module, these two modules should be trained separately. Therefore, the training process is divided into three stages: Gridnet module training stage, refinement module training stage and jointly training stage.

- Firstly, only the Gridnet module is trained. We concatenate the 3-channel 320×320 image patch and corresponding trimap patch along the channel to form the 4-channel 320×320 input. We feed the Gridnet with the input and pixel-wise add the output of the Gridnet with trimap patch to form the initially predicted alpha matte. All the training strategies proposed previously are used. We initialize the Gridnet module with Xavier random variables. The optimizer is Adam [30] with learning rate 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$. The mini-batch size is 8. We trained about 200 epoches before the Gridnet is converged. The loss function is the overall loss function $L_{overall}$ defined by Equ. (5.1).

- After the Gridnet module is converged, we fix the parameters in this module and attach the refinement module. We concatenate the initially predicted alpha matte with the original 3-channel 320×320 image patch and feed them to the refinement network. Then, the output of the refinement network is pixel-wise added with the initially predicted alpha matte to form the refined alpha matte. The refinement module is also initialized by Xavier random variables. The solver is also Adam [30] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$. Learning rate is 10^{-4}. The mini-batch

size is 8. We trained about 50 epoches before the refinement module is converged. All the training strategies proposed previously are also used. Only the alpha prediction loss function defined by Equ. (5.2) is used.

$$\mathrm{L1}^{alpha}(\alpha_{pre}, \alpha_{gt}) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\sqrt{(\alpha_{pre}(i,j) - \alpha_{gt}(i,j))^2 + \sigma^2}. \tag{5.2}$$

- After the refinement module is also converged, the whole network is trained jointly with learning rate 10^{-5} until it is converged. The loss function is still alpha prediction loss function defined by Equ. (5.2).

When testing, we concatenate the original resolution image and its trimap to form a 4-channel input. The input is passed forward to the Gridnet module and output the initially predicted alpha matte. Then the initially predicted alpha matte is concatenated with the original image to be forward propagated through the refinement module to output the refined alpha matte.

5.3 Evaluation Criteria

There are four error metrics used as the criteria in evaluating the matting performances. The first one is SAD, the sum of absolute differences. It measures the differences between unknown region of alpha mattes by taking the absolute difference between each pixel in the unknown region of the ground truth alpha matte and the corresponding pixel in the unknown region of the predicted alpha matte. These differences are summed to create a metric of alpha matte differences. The SAD takes the formula as:

$$\mathrm{SAD}(\alpha_{pre}, \alpha_{gt}) = \sum_{i}\Big|\alpha_{pre}(i) - \alpha_{gt}(i)\Big|, i \in U, \tag{5.3}$$

where $|\cdot|$ denotes the absolute value, U is the unknown region, $\alpha_{gt}(i)$, $\alpha_{pre}(i)$ denotes the alpha value of i in the ground truth and the predicted alpha matte, respectively. The lower the SAD function value, the better the matting result.

The second error metric is MSE, the mean squared error. It measures the average of the squared difference between the ground truth alpha matte and the predicted alpha matte. It takes the following formula:

$$\text{MSE}(\alpha_{pre}, \alpha_{gt}) = \frac{1}{M} \sum_{i} \Big[\alpha_{pre}(i) - \alpha_{gt}(i)\Big]^2, i \in U, \tag{5.4}$$

where U is the unknown region, M is the number of pixels in the unknown region of trimap. The lower the MSE error value, the better the matting performance.

The above two error metrics are common criteria in calculating the differences between images, however, it is not enough to evaluate matting performance by only using these two error metrics. Because these two measures are not always correlated to the visual quality, perceived by a human observer. Hence, some kind of perceptual error metrics are needed to correctly reflect the visual differences between two images, and to realize a fair perceptual comparison of different matting methods.

Therefore, gradient and connectivity error functions are proposed by [47] as perceptual error metrics. Gradient error metric prefers the predicted alpha matte with the same gradient value at each pixel as the ground truth alpha matte. It punishes over-smoothness and erroneous discontinuities in the alpha matte. As Figure 5.1 shows, the matting result in (c) has higher visual quality than the matting result in (d). However, the SAD error is lower in (d) than (c), and the gradient error is lower in (c) than (d). Hence, gradient is a more reasonable error criterion than SAD/MSE error in visual quality comparison.

The gradient error function is given by:

$$\text{Grad}(\alpha_{pre}, \alpha_{gt}) = \sum_{i} \Big[||\nabla\alpha_{pre}(i)|| - ||\nabla\alpha_{gt}(i)||\Big]^2, i \in U, \tag{5.5}$$

where U is the unknown region, $||\nabla\alpha_{pre}(i)||$ and $||\nabla\alpha_{gt}(i)||$ are the amplitude of the gra-

(a) Image (b) Zoom (c) SAD:1215 (d) SAD:806

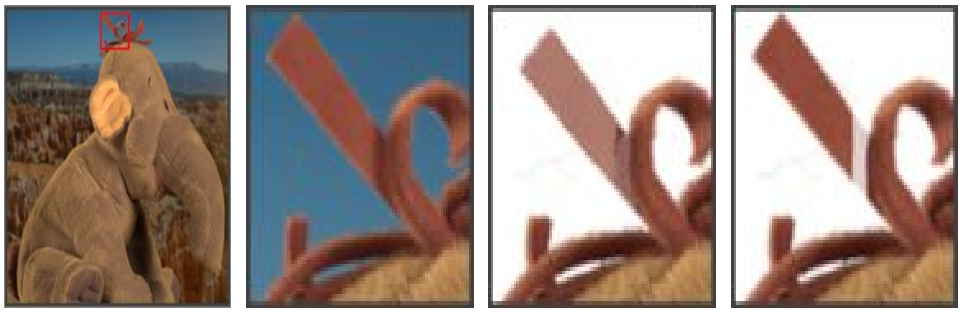
Figure 5.1: The example indicates that gradient error metric is needed and even more suitable than SAD error metric in evaluating the visual quality of extracted foreground objects. The images and data are cited from [47].

dient, which can be calculated by:

$$\begin{aligned} ||\nabla\alpha_{pre}(i)|| &= \sqrt{\frac{\partial\alpha_{pre}(i)}{\partial x}^2 + \frac{\partial\alpha_{pre}(i)}{\partial y}^2}, \\ ||\nabla\alpha_{gt}(i)|| &= \sqrt{\frac{\partial\alpha_{gt}(i)}{\partial x}^2 + \frac{\partial\alpha_{gt}(i)}{\partial y}^2}. \end{aligned} \tag{5.6}$$

Here, $\nabla\alpha_{pre}(i)$ is the normalized gradient of the predicted alpha matte at pixel i and $\nabla\alpha_{gt}(i)$ is the normalized gradient of the ground truth alpha matte at pixel i. Gradients are calculated by convolving the alpha mattes with first order Gaussian derivative filters, whose variance is $\sigma = 1.4$. By using a 2-D Gaussian kernel, derivatives along x and y directions are calculated. The lower the gradient error, the more similar the predicted alpha matte with the ground truth alpha matte.

Connectivity error function punishes the disconnected foreground objects, such as a human with a disconnected piece of hair floating in the air. It is an error metric, which is sometimes better than SAD/MSE in visual quality assessment. Figure 5.2 shows the situation that the foreground object has a lower SAD value (i.e. (d)) but with worse visual effect, because of the higher connectivity error value.



(a) Image (b) Zoom (c) SAD:312 (d) SAD:83

Figure 5.2: The example indicates that connectivity error metric is needed and even more suitable than SAD error metric in evaluating the visual quality of extracted foreground objects. The images and data are cited from [47].

The connectivity error is defined by:

$$\text{Conn}(\alpha_{pre}, \alpha_{gt}) = \sum_i \Big| f\big(\alpha_{pre}(i), \Omega\big) - f\big(\alpha_{gt}(i), \Omega\big) \Big|, i \in U, \tag{5.7}$$

where $f(\cdot)$ measures the degree of connectivity from pixel i to a source region Ω, in predicted alpha matte and in ground truth alpha matte, respectively. $\alpha(i)$ is the alpha value of pixel i. U is the unknown region.

All of the above four error metrics will be used in our experiments to quantitatively evaluate our matting performance and provide a fair comparison with other methods.

5.4 Testing Dataset

We evaluated our method on the Composition-1k testing dataset [65], which is a challenging, high-quality test set. It consists of 1000 images, generated by compositing each of 50 foreground object onto 20 background images based on the alpha matte ground truth and matting model described by Equ. (1.1).

The testing dataset has the following properties: First of all, they are in a natural setting, with natural background, natural foreground and natural lighting. Secondly, the

foreground objects have wide variety, and cover many difficult matting cases, such as human with long hair, animals with downy fur, transparent glass, multi-hole web. Thirdly, they cover many conditions difficult for existing alpha matting methods, such as highly textured backgrounds and overlapped foreground and background color distributions. Fourthly, the testing images are in high resolutions (e.g., 1920×1762), no one has the resolution less than $1K$. All these characteristics reveal problems of existing matting methods, and make the Composition-1k testing dataset a fair, representative and challenging dataset in evaluating alpha matting performance.

5.5 Experimental Results

5.5.1 Measurement

Our matting performances are quantitatively evaluated by the four error metrics described in Section 5.3 : SAD, MSE, gradient error and connectivity error. We normalized both the ground truth alpha matte and the predicted alpha matte to 0 to 1 when calculating SAD, MSE and connectivity error metrics. Each error value is the average value of the 1000 testing images. The trimap has divided the image into the definite foreground regions, the definite background regions and the unknown regions clearly. We only focus on the matting quality in the unknown regions. Hence, all the error metrics are calculated only within the unknown regions.

5.5.2 Evaluation of Gridnet Configuration

First of all, we test the configurations of Gridnet by using a Gridnet with 5 rows, 2 subsampling columns followed by 2 upsampling columns. The number of feature maps from the first row to the fifth row are: 16, 32, 64, 128, 256. We change some parts of our network

configurations, compare the matting results in order to find the best configurations. In Table 5.1, we show the results of Gridnets with different configurations on the Composition-1k testing dataset. "Fusion" represents the way feature maps fused at the intersection of the Gridnet (i.e. between horizontal and vertical blocks). "Add" means the output feature maps of the horizontal block and the output feature maps of the vertical block are added to form the final feature maps at the intersection (e.g. $X_{i,j}$). "Concat" means the two output feature maps are concatenated along the channel, which significant increases the number of feature maps at the intersection of the Gridnet. As the matting results indicate, "Concat" degrades the Gridnet performance. "h-residual" indicates the horizontal block (i.e. green block) is a residual block, and "v-residual" indicates the vertical block is also a residual block. The results indicate that adding the residual connection in the vertical block is unnecessary, which is not proved to be an optimal choice. The proposed configurations are marked in bold. We prefer the Gridnet with "Add" as fusion method, and only design horizontal computation block as residual block.

Configuration			Quality Assessment			
Fusion	h-residual	v-residual	SAD	MSE	Gradient (10^3)	Connectivity (10^3)
Add	✓		**51.448**	**0.014**	31.033	**51.082**
Add	✓	✓	52.316	0.015	**30.704**	51.471
Concat	✓		54.912	0.017	35.098	56.320

Table 5.1: Evaluation of different configurations. The best results are emphasized in bold.

5.5.3 Evaluation of Structure

As we mention in Section 4.1.2, our Gridnet has higher flexibility in performance by adjusting the number of rows, columns, and feature maps in the first row of Gridnet. In our network structure evaluation stage, we change those numbers, in order to explore the

impact of the numbers and find the best combination of numbers.

Structure			Quality Assessment			
Rows	Columns	Feature maps	SAD	MSE	Gradient (10^3)	Connectivity (10^3)
6	6	16	52.630	0.017	**30.921**	52.683
5	**4**	**16**	**51.448**	**0.014**	31.033	**51.082**
4	4	16	60.182	0.022	33.456	60.709

Table 5.2: Evaluation of different network structures. The first column of the table is the number of the rows (r), the second column is the number of the columns (c) and the third column is the number of feature maps in the first row (f). The best results are emphasized in bold.

In our experiment, according to our hardware constraints, we tried different combinations of numbers, including: $6r6c16f$, $6r4c16f$, $6r2c16f$, $5r6c16f$, $5r4c16f$, $5r2c16f$, $4r6c32f$, $4r6c16f$, $4r4c16f$. We only record the best result for each different row number. As Table 5.2 describes, the Gridnet with 5 rows 4 columns, 16 feature maps in the first row ($5r4c16f$) has the best performance in our matting task, based on our training dataset and hardware constraints. When our Gridnet has $5r4c$, increasing the number of rows or columns, respectively or together, will not be proved to improve the performance while increasing the training complexity. However, when the number of rows is less than 5, the matting performance degrades, even if we increase the column number and the feature map number in the first row. Hence, we choose Gridnet with 5 rows 4 columns and 16 feature maps in the first row as our network structure.

5.5.4 Performance Comparison with Refinement Module

In this section, we will explore the influence of the addition of refinement module. As we introduced in the Section 4.3, we used two different network architectures as the refinement

networks in refinement modules. We hope to use the refinement module to supplement the detail information missed by our initial alpha prediction, in order to improve our matting results. Table 5.3 shows the quality assessment of matting results, when we do not add any refinement module, add the four conv layers as the refinement network in refinement module, or add the Gridnet with two rows and four column as the refinement network in refinement module.

Structure	SAD	MSE	Gradient (10^3)	Connectivity (10^3)
Gridnet	**51.448**	**0.014**	**31.033**	**51.082**
Gridnet+Four conv layers	51.506	0.015	31.713	52.057
Gridnet+Gridnet (2r4c16f)	51.395	0.015	31.316	51.898

Table 5.3: Performance comparison with different refinement module. The best results are emphasized in bold.

The intention of us to add the refinement module is to overcome the over smoothness problem existed at the edge of foreground objects. As Table 5.3 shows, these two network architectures do not have this effect. The reason may have two aspects: 1) The main Gridnet structure may have learnt the edge information of the image, those pieces of information can not be improved by added network. 2) The network architecture of refinement module is not suitable for its task. Since our main network has achieved a pretty good matting result, we discard the refinement module and use the Gridnet with ($5r4c16f$) as our final network architecture.

5.5.5 Performance Comparison with Other Matting Methods

In this section, we will compare the matting results of our method with other state-of-the-art matting methods quantitatively and visually.

The compared matting methods in Table 5.4 including 10 different matting methods. Among them, shared sampling matting [14], comprehensive sampling matting [53] and global sampling matting [21], are representatives of sampling-based matting methods. Closed-form matting [34] and KNN matting [6] are "local" and "nonlocal" representatives of propagation-based matting methods. Learning based matting [68] solves matting problem with knowledge of machine learning. DCNN matting [7], GAN matting [38] and deep image matting [65] are emerging matting methods based on deep learning.

Table 5.4 shows the quantitative comparison of our results and other matting methods on the Composition-1k testing dataset. We should notice that, the deep learning matting method does not publish their code. We reproduce their method, according to the network architecture and training implementation described by them in their paper. However, since parts of the original training dataset is not provided by them, we can not get the same matting results as they stated in their paper. To fair comparison, we quote the matting results released by themselves in their paper. In the same way, we quote the matting results provided by GAN matting method [38].

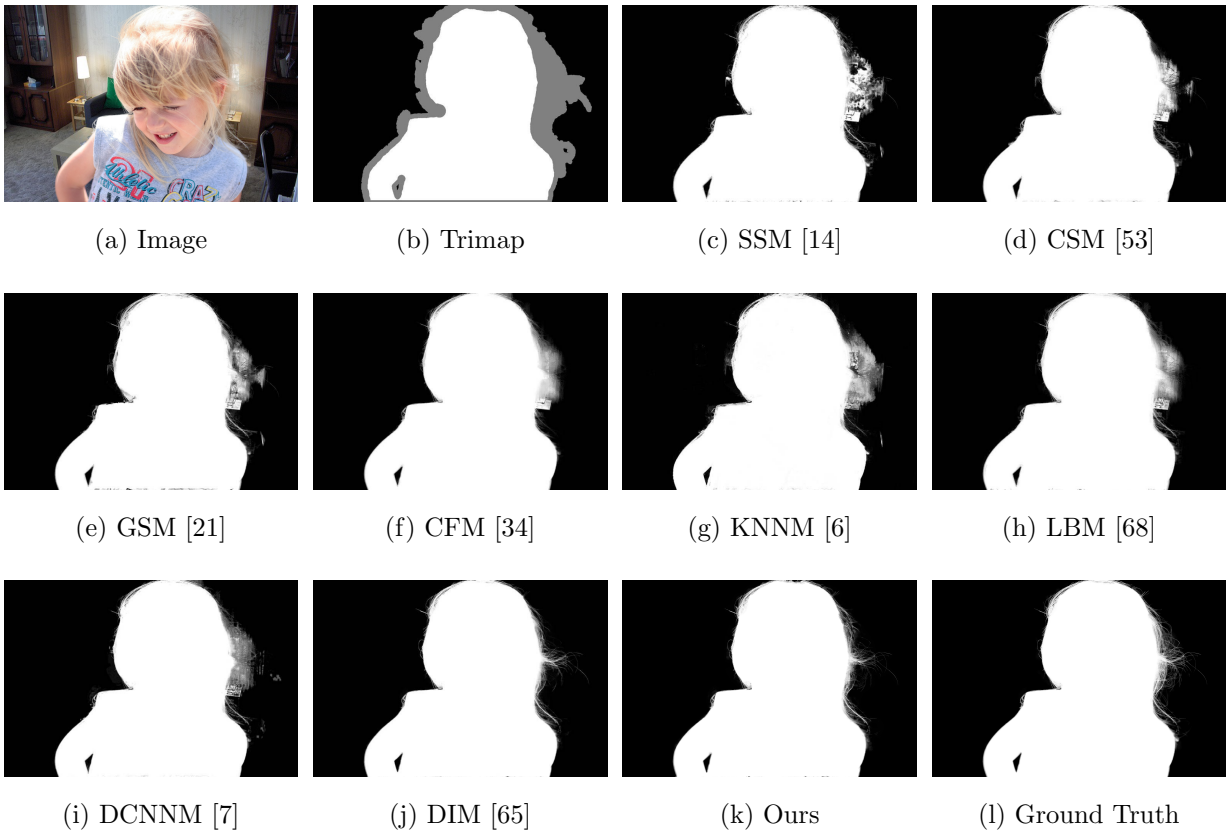
As Table 5.4 shows, our method performs much better than the other matting methods except for the deep image matting with refinement method [65], which currently is the best matting method in natural image matting. However, the deep image matting method has $49,300$ images as training dataset, in contrary, we only have $43,100$ images for training. In addition, the network of deep image matting has 26 million trainable parameters before adding refinement stage, which is more than three times of the number of parameters in our Gridnet architecture. The number of trainable parameters can prominently affect computation complexity and memory usage. With the same experiment environment, the more the parameters the longer the training time. Our matting method trades off the computation complexity, memory usage and matting performance well.

The visual comparisons with deep image matting and other image matting methods are shown in Figure 5.3, Figure 5.4, Figure 5.5 and Figure 5.6, which demonstrate the different

Methods	SAD	MSE	Grad(10^3)	Conn(10^3)
Shared Sampling Matting (SSM) [14]	128.9	0.091	126.5	135.3
Comprehensive Sampling Matting (CSM) [53]	143.8	0.071	102.2	142.7
Global Sampling Matting (GSM) [21]	133.6	0.068	97.6	133.3
Closed-Form Matting (CFM) [34]	168.1	0.091	126.9	167.9
KNN Matting (KNNM) [6]	175.4	0.103	124.1	176.4
Learning Based Matting (LBM) [68]	113.9	0.048	91.6	122.2
DCNN Matting (DCNNM) [7]	161.4	0.087	115.1	161.9
GAN Matting (GANM) [38]	68.8	0.032	51.4	60.2
Deep Image Matting (without refinement) [65]	54.6	0.017	36.7	55.3
Deep Image Matting (DIM) [65]	**50.4**	**0.014**	**31.0**	**50.8**
Ours	51.4	**0.014**	**31.0**	51.0

Table 5.4: Quantitative comparison of various matting methods on the Composition-1k dataset. The best results are emphasized in bold. "Grad" means Gradient, "Conn" means Connectivity. The results are from [38] and [65].

performance between our method and others. Because the code of deep image matting method are not public, we compare our visual results with the same ones provided by [65].



(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

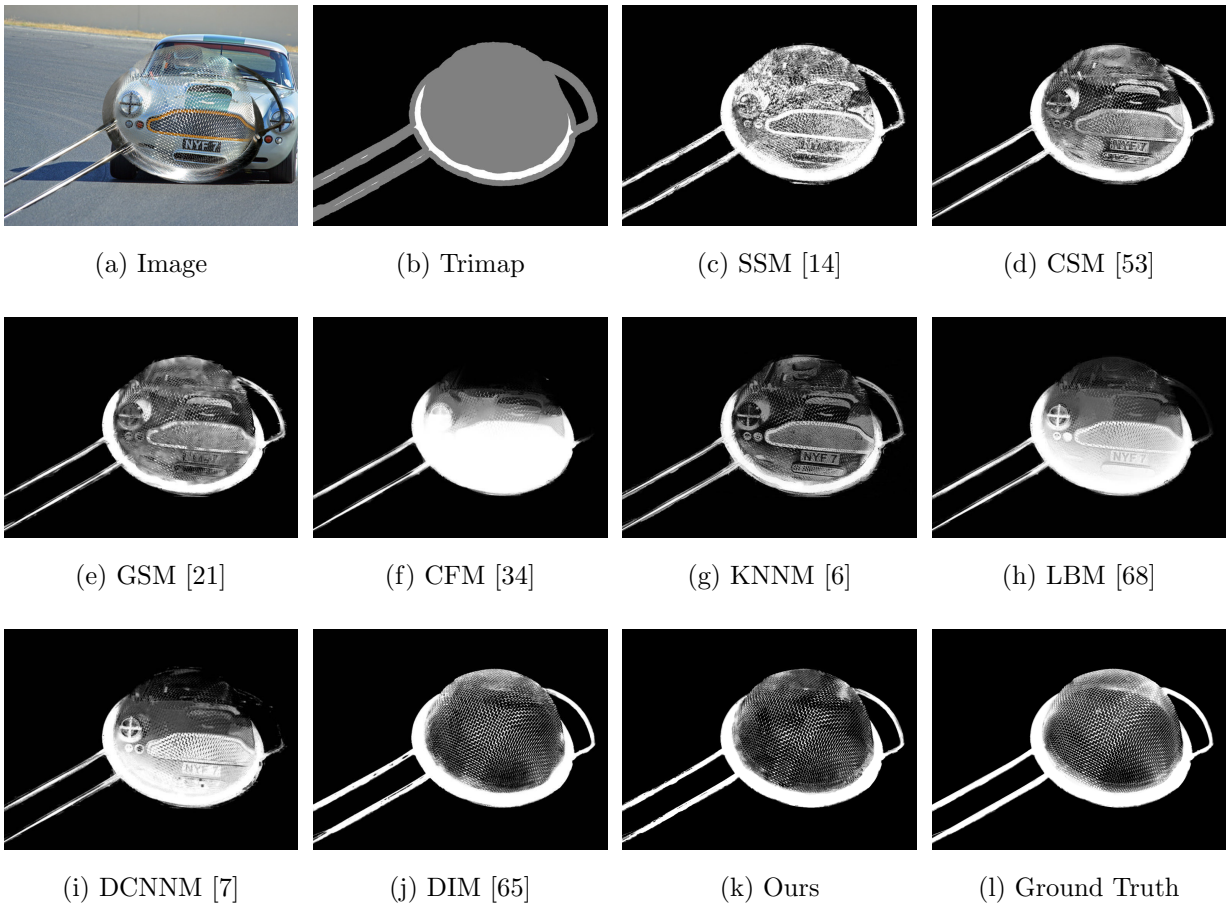
(e) GSM [21] (f) CFM [34] (g) KNNM [6] (h) LBM [68]

(i) DCNNM [7] (j) DIM [65] (k) Ours (l) Ground Truth

Figure 5.3: Visual comparison on *Girl* with previous matting methods.

In Figure 5.3, the light of the lamp in the background passes through the hair of the small girl in the foreground, which seriously effects the matting performance of this area. As we can see in this figure, previous matting methods (except for deep image matting [65]) can not deal with this issue well. With the bad influence of the lamp light, the hair floating in the air are in a mess in the predicted alpha mattes produced by previous matting methods. Only our method and deep image matting method [65] can extract the hair clearly and clean. However, both of us inevitably loss some detail of the hair at that area.

Figure 5.4 shows the situation where the foreground is a mesh with fine micro-holes. The light of background image passes through every micro-hole. The conventional matting



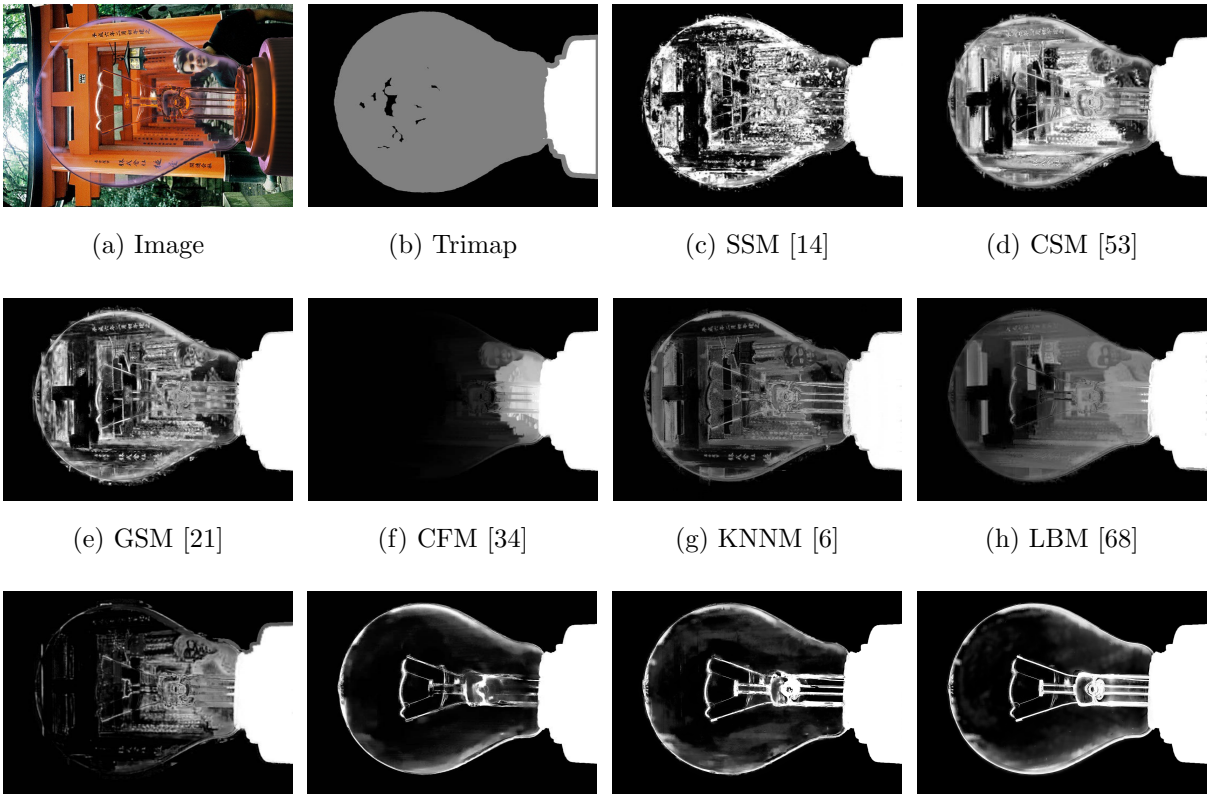
(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) GSM [21] (f) CFM [34] (g) KNNM [6] (h) LBM [68]

(i) DCNNM [7] (j) DIM [65] (k) Ours (l) Ground Truth

Figure 5.4: Visual comparison on *Strainer* with previous matting methods.

methods, which distinguish the foreground from background relying on the color or position information or some propagation rules, inevitably define the background as foreground and retain background information in alpha mattes by mistake. Our method and deep image matting method [65] have the ability to comprehend the semantic information and context information of the image. For example, with the training, our network may "know" the foreground is a "grid". Hence, our matting result only retains the grid frame structure as foreground object and discards the color information in the micro-holes.



(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) GSM [21] (f) CFM [34] (g) KNNM [6] (h) LBM [68]

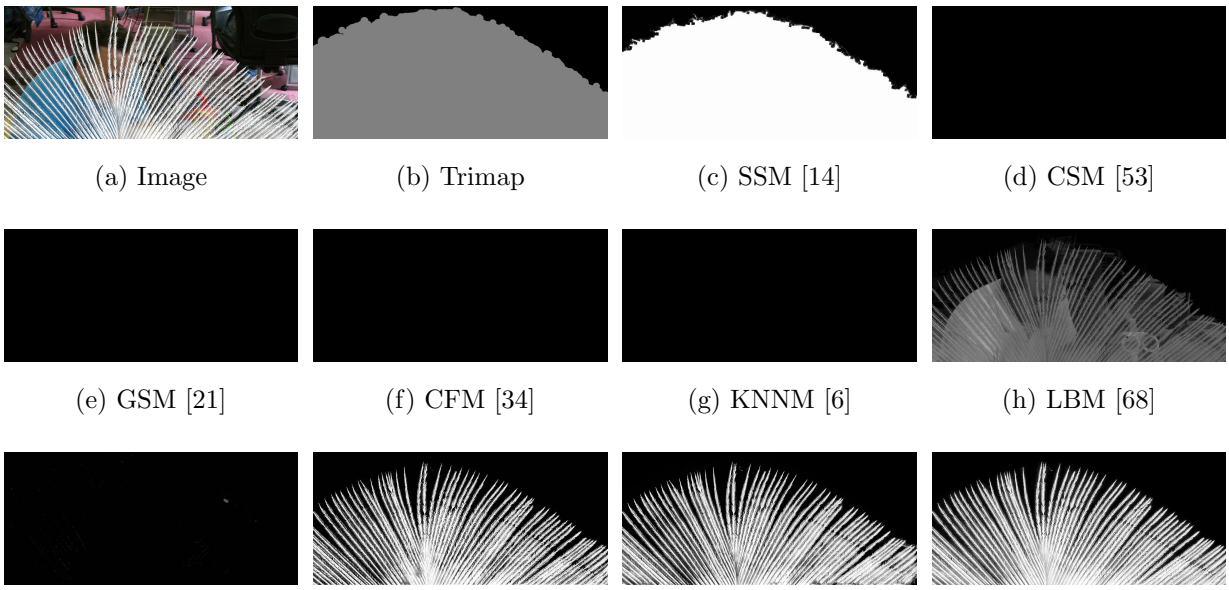
(i) DCNNM [7] (j) DIM [65] (k) Ours (l) Ground Truth

Figure 5.5: Visual comparison on *Bulb* with previous matting methods.

Figure 5.5 provides the situation where the foreground is a transparent item. As the figure shows, all the previous matting methods judge the background as foreground and retain the background information in their predicted alpha mattes by mistake. Only our method and deep image matting method [65] discard the background information successfully by "learning" the semantic information of this image, which may indicate that, the

foreground is a transparent "bulb".

In Figure 5.6, there is no pure foreground pixels in the image. The totally dark alpha mattes show the helplessness of conventional matting algorithms in this situation. Because they can neither find the suitable samples to represent the true foreground color of the unknown pixel, nor find the weight between pixels to propagate the alpha value. The learning based matting method [68] can solve this problem to a certain extent. But the background information still wrongly existed, and the alpha matte is over smooth. Our matting method and deep image matting [65] can extract the foreground object clearly and purely, with comprehending the semantic information and context information of the image.



(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) GSM [21] (f) CFM [34] (g) KNNM [6] (h) LBM [68]

(i) DCNNM [7] (j) DIM [65] (k) Ours (l) Ground Truth

Figure 5.6: Visual comparison on *Plume* with previous matting methods.

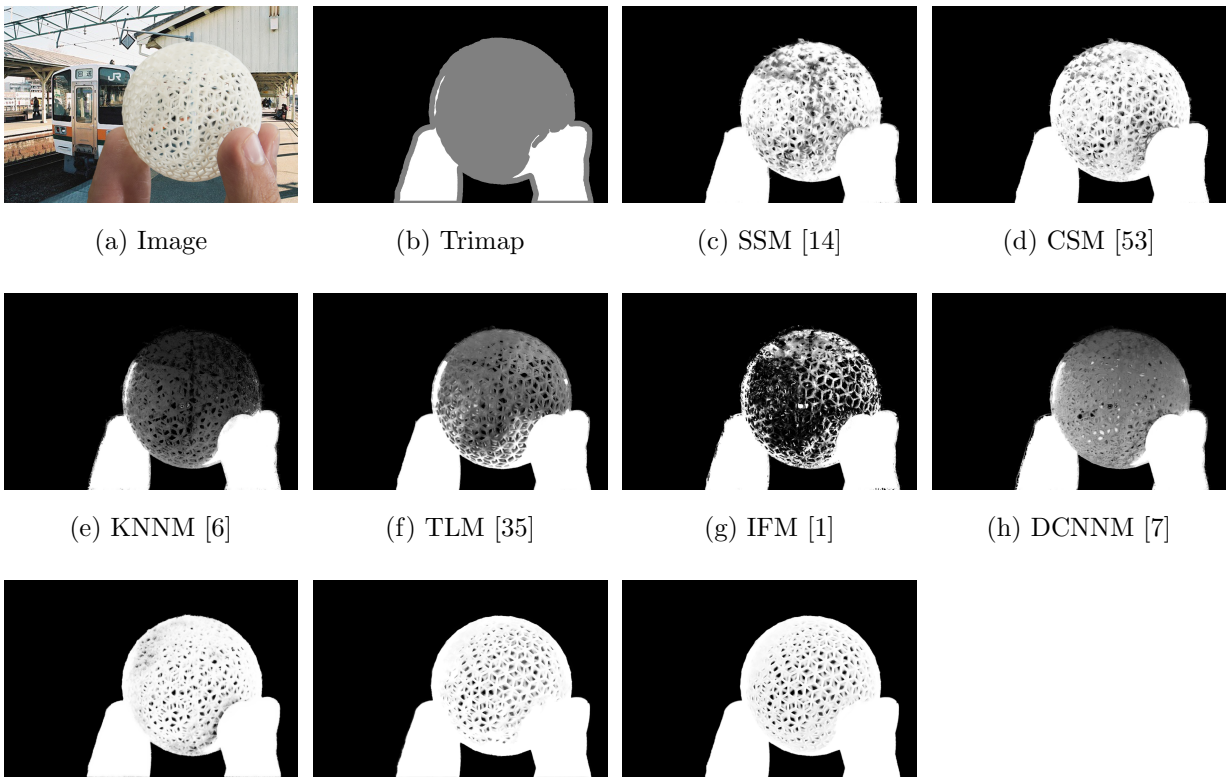
After comparing our matting method with deep image matting [65] and some previous state-of-the-art matting methods, we focus on the comparison between our matting method and the GAN matting method [38], which is a newest published matting method based on deep learning, having top matting performance. Because the GAN matting method [38]

neither provides the original code of their method nor provides matting results on the same images that DIM [65] provides. We have to compare our visual results with the ones provided by [38].

The visual comparison is shown from Figure 5.7 to Figure 5.12. The matting algorithms involved in the comparison include: shared sampling matting method [14], comprehensive sampling matting method [53], KNN matting method [6], three-layer matting method [35], information flow matting method [1], DCNN matting method [7] and GAN matting method [38]. Among them, shared sampling matting method [14] and comprehensive sampling matting method [53] are representatives of sampling-based matting methods, KNN matting method [6] is the representative of the propagation-based matting methods, information flow matting method [1] is the currently best conventional matting method, and GAN matting method [38] is the newest published matting method based on deep learning.

As the Figure 5.7 shows, our matting result is much better than the one produced by the GAN matting method [38]. The GAN predicted alpha matte, is lack of stereoscopic sense, and is blurry in the area with holes. It seems to ignore the specific structure of the ball and just keep the general appearance. In addition, the result from information flow matting method [1] shows apparent discontinuity in the edge and the inside of foreground object. Its performance is worse than the Three layer matting method [35]. However, the information flow matting method performs much better than the Three layer matting method on the online benchmark [47], which contains only 8 images as testing dataset. This demonstrates the point we made in Chapter 4 that performing well on the online benchmark does not mean having a good result in dealing with a natural image, because of the inherently unrealistic nature of the dataset.

In Figure 5.8, we shows our matting performance in dealing with a plant, since plants are inevitably appeared in natural images. We choose a plant with white pappi, which increases the difficult in matting. As Figure 5.8 shows, we retain the detail of the pappi clearly,

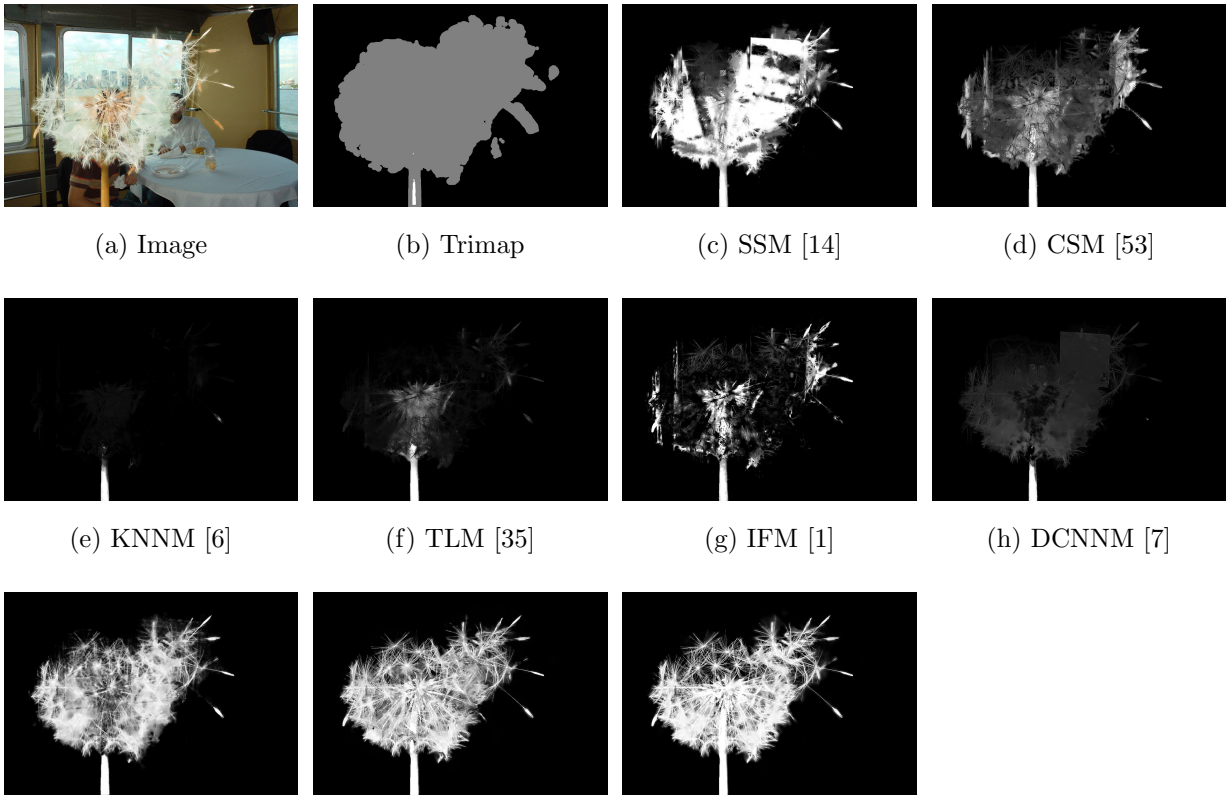


(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) KNNM [6] (f) TLM [35] (g) IFM [1] (h) DCNNM [7]

(i) GAN [38] (j) Ours (k) Ground Truth

Figure 5.7: Visual comparison on *Ball* with previous matting methods.



(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) KNNM [6] (f) TLM [35] (g) IFM [1] (h) DCNNM [7]

(i) GAN [38] (j) Ours (k) Ground Truth

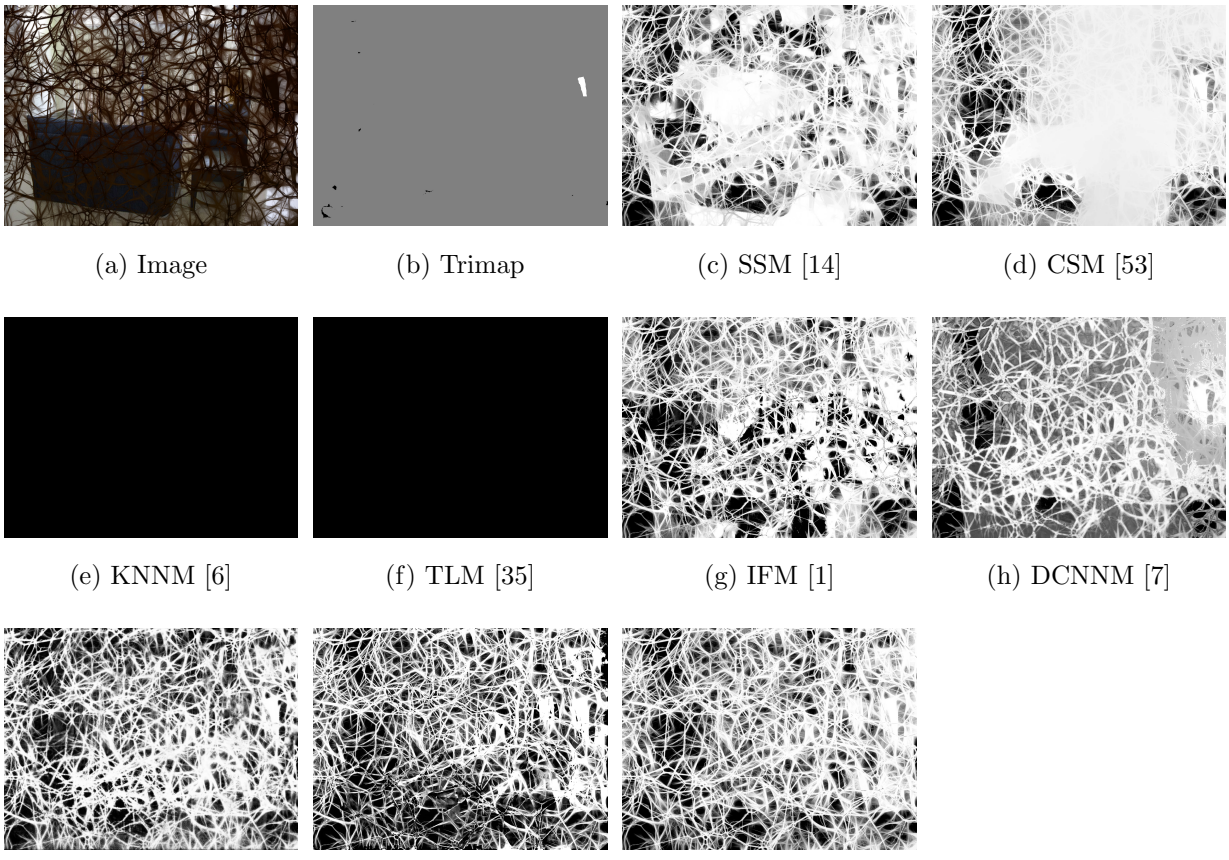
Figure 5.8: Visual comparison on *dandelion* with previous matting methods.

however, the matting result predicted by GAN is blurry and the stem of the dandelion is judged as background composition by mistake. Other matting methods perform bad in extracting the hair-like, white pappi.

In Figure 5.9, we expand our experiment further to deal with the image with fine tangled fibers as foreground objects. It is really difficult for human beings to define the absolute foreground pixels and absolute background pixels of the image, so the trimap is nearly grey. Extracting the fibers may be considered as useless, but an accurate alpha matte can help us to recover the background items, which is occluded by foreground objects. This may provide assistance in medicine and criminal investigation. As we can seen in Figure 5.9, our matting method can extract the fiber information clearly. However, if we magnify the alpha matte predicted by GAN, we can see the serious discontinuity in fibers and over smoothness at the edge of each fiber. The KNN matting method [6] and the Three layer matting method [35] fail to deal with this kind of images.

In Figure 5.10, we show our matting method in dealing with animals with hairs. When we pay attention to horse's whiskers, hairs and even eyelash, we can see that our method retains all the detail information, which are lost to various degrees by other matting methods. In addition, the horse's hair on its back is black, which is the same as the color of the window frame in the background. All the conventional matting methods and the DCNN matting method [7] define the window frame, in the unknown region of the trimap, as part of the foreground object. This mistake is inevitably generated, when the matting methods use the color information to extract foreground objects from images, or when they use the color information to define the propagated weight between pixels.

Figure 5.11, Figure 5.12 and Figure 5.13 show our better matting performance in dealing with transparent or semi-transparent foreground objects. In fact, failing in extracting transparent or semi-transparent foreground objects has plagued researchers for a long time. However, as we mention in our thesis repeatedly, conventional matting methods just rely on color and spatial position information as the distinguishing features, and cannot com-

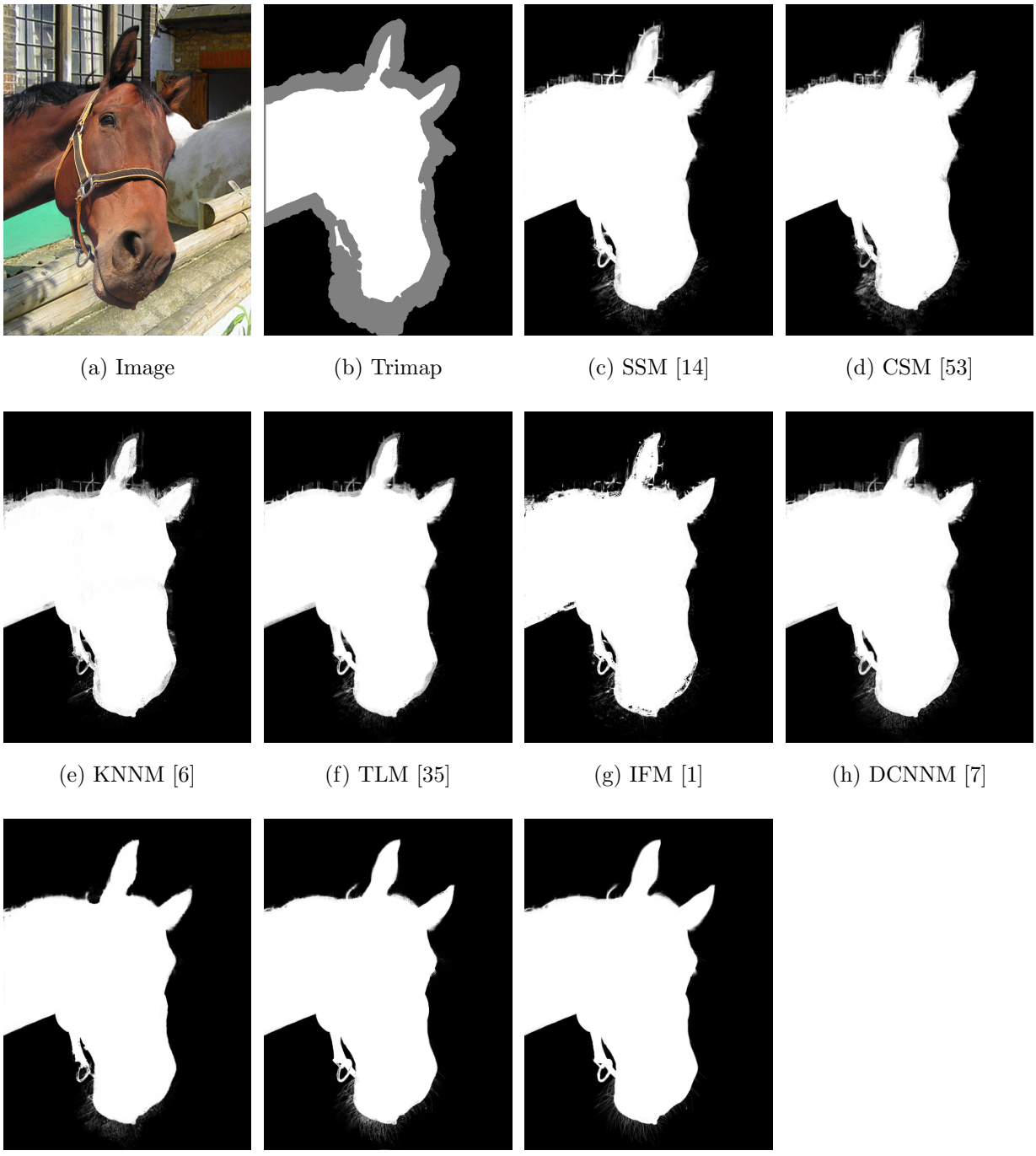


(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) KNNM [6] (f) TLM [35] (g) IFM [1] (h) DCNNM [7]

(i) GAN [38] (j) Ours (k) Ground Truth

Figure 5.9: Visual comparison on *network* with previous matting methods.



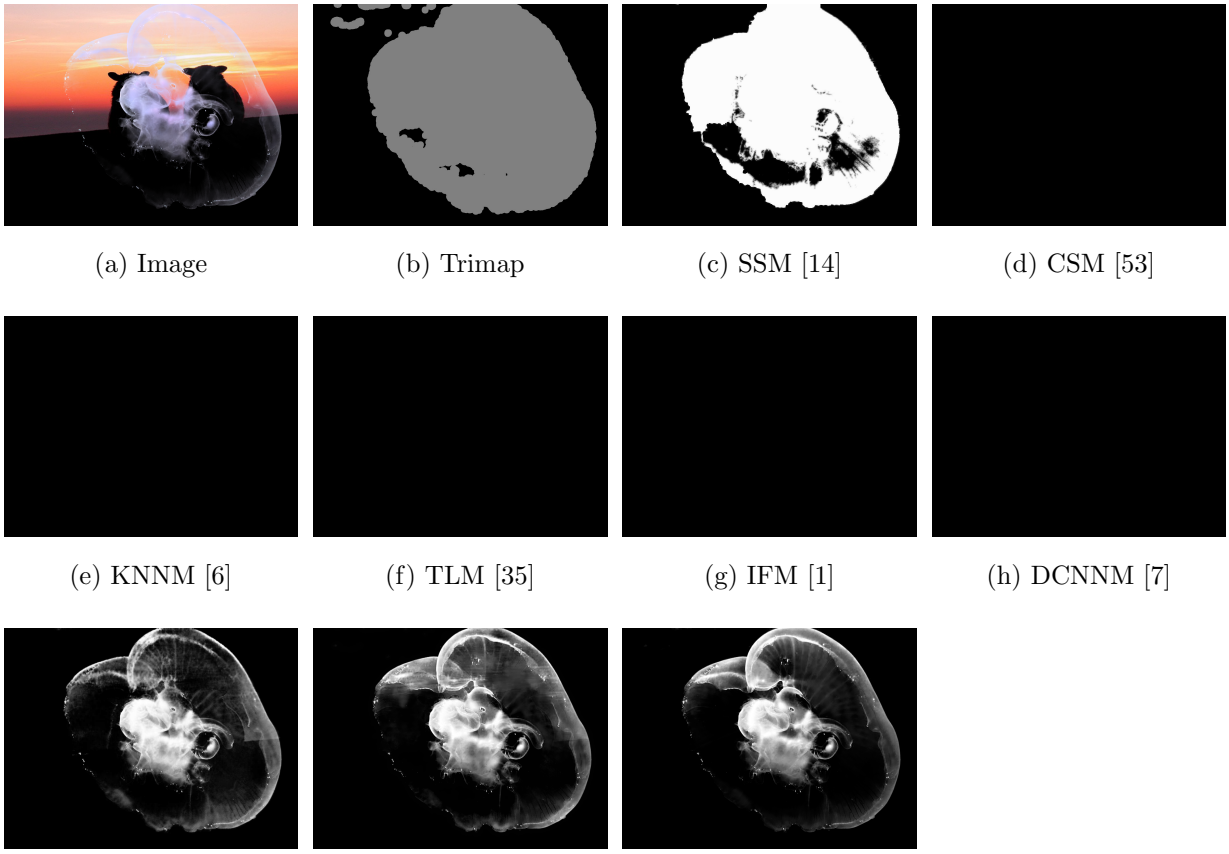
(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) KNNM [6] (f) TLM [35] (g) IFM [1] (h) DCNNM [7]

(i) GAN [38] (j) Ours (k) Ground Truth

Figure 5.10: Visual comparison on *horse* with previous matting methods.

prehend semantic information of images. This inevitably makes the conventional matting methods fail in dealing with those kinds of matting problems. These figures indicate the tremendous advantage of deep learning based matting methods in dealing with these kinds of matting problems. In addition, our matting results are much more realistic, smooth than the GAN matting method [38].



(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) KNNM [6] (f) TLM [35] (g) IFM [1] (h) DCNNM [7]

(i) GAN [38] (j) Ours (k) Ground Truth

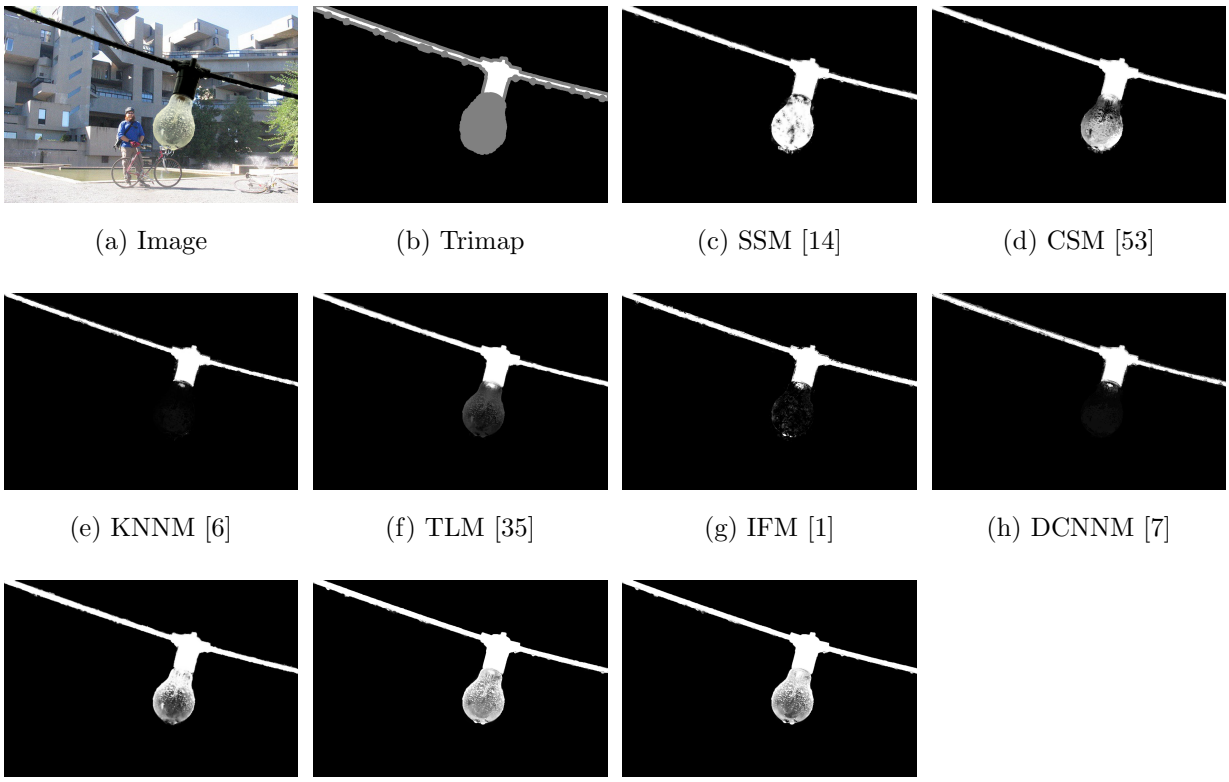
Figure 5.11: Visual comparison on *jellyfish* with previous matting methods.

(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) KNNM [6] (f) TLM [35] (g) IFM [1] (h) DCNNM [7]

(i) GAN [38] (j) Ours (k) Ground Truth

Figure 5.12: Visual comparison on *glass* with previous matting methods.



(a) Image (b) Trimap (c) SSM [14] (d) CSM [53]

(e) KNNM [6] (f) TLM [35] (g) IFM [1] (h) DCNNM [7]

(i) GAN [38] (j) Ours (k) Ground Truth

Figure 5.13: Visual comparison on *pexel* with previous matting methods.

Chapter 6

Conclusions and Discussions

In this section, we will conclude our matting method and discuss some doubtful point existed in our network structure.

6.1 Conclusions

Extracting foreground objects from a natural image and calculating the alpha matte of the image are important requirements for various applications, such as virtual reality, digital image/video editing or segmentation. The development of digital cameras and films with virtual scene puts forward higher requirements for matting techniques.

Conventional matting algorithms for natural image matting (alpha matting) have developed for more than 30 years, and they have achieved satisfactory matting results in many cases. However, due to their inherent characteristics, there are still many matting cases are difficult for conventional matting algorithms to deal with. For example, when the foreground objects of natural images are hairs, feathers, meshes, fibers, transparency/semi-transparency or somethings with no pure foreground pixels, the matting performance of

conventional algorithms will be severely degraded. Fortunately, deep learning techniques provide new possibilities for solving alpha matting problems.

In this thesis, we proposed a novel matting method based on deep learning. By designing a residual convolutional neural network: Grid network, we compute the alpha matte directly from the original RGB image and its trimap. Instead of relying on color information, our network can extract the semantic information and the context information of the image. As the experimental results show in Chapter 5, by learning the natural structure and extracting the feature of the images, our matting method can compute accurate alpha mattes even in the difficult cases we mentioned above.

Our network architecture is a Gridnet with 5 rows, 4 columns and 16 feature maps in the first row. Unlike other network structures, our network is extremely flexible. This flexibility is reflected in the following two aspects:

- Our network has adaptivity, because it can choose paths to pass the information by itself. Our network is a 2-D grid network, the horizontal computation blocks are residual convolutional blocks without upsampling or downsampling, and the vertical computation blocks are upsampling/downsampling convolutional blocks. When the information just passes through the horizontal computation blocks, the details of the images are reserved. When it passes through the vertical computation blocks, the high-level abstract information can be extracted from the images. In our method, we do not assign the path, along which the information is passed. Hence, our network can choose any path to pass the information according to requirements. As the matting results show in Chapter 5, after training, our network either retains the rich details or extracts abstract semantic information from the image. This indicates that our network can choose its own information transmission well.

- Our network has scalability. With changing the number of rows or columns of the network, or changing the number of filters in every rows, our network has the ability

to fit the size of dataset to a large extent. In Chapter 5, we changed the number of rows, columns and feature maps in the first row of our Gridnet to find the suitable number combination for our dataset and matting task. Gridnet with $5r4c16f$ is the most suitable structure for current size of dataset. However, if the matting dataset is enlarged, we believe our Gridnet can also be suitable by adjusting the numbers of the rows, columns and feature maps. In contrast, all the other existed matting methods based on deep learning may face challenges if a larger and more diverse dataset appears, because of their fixed network structures.

The experimental results demonstrate that our proposed matting method outperforms most previous matting methods and is comparable to the best matting method: deep image matting method [65]. Considering that we only use less than one-third parameters of deep image matting, our method achieves a remarkable result in balancing the calculation complexity and matting performance.

6.2 Discussions

In this section, we discuss the reason why adding the number of rows of Gridnet does not lead to a better matting result. As Table 5.2 shows in Chapter 5, the matting results generated by Gridnet with $6r6c16f$ is not better than the results generated by Gridnet with $5r4c16f$. This phenomenon is beyond our expectation of the Gridnet. In fact, we believed that the bigger the Gridnet, the better the matting results, before our experiments.

Here, we discuss the possible causes of this situation. As we can see in our Gridnet with $6r6c$ (Figure 6.1), the path connected input and output is shorter across the high resolution stream (red path in Figure 6.1) than with the low resolution ones (blue path in Figure 6.1). The longer the path, the deeper the network. However, deeper network may lead to vanishing gradients and are more difficult to train. In our matting task, the

information passing through the blue path may lose because of the vanishing gradients. Or, the blue path is not selected by network to avoid overfitting, because of the limited number of training dataset. Due to those possibilities, the Gridnet with $5r4c16f$ is the most suitable network architecture for our matting task and training dataset.

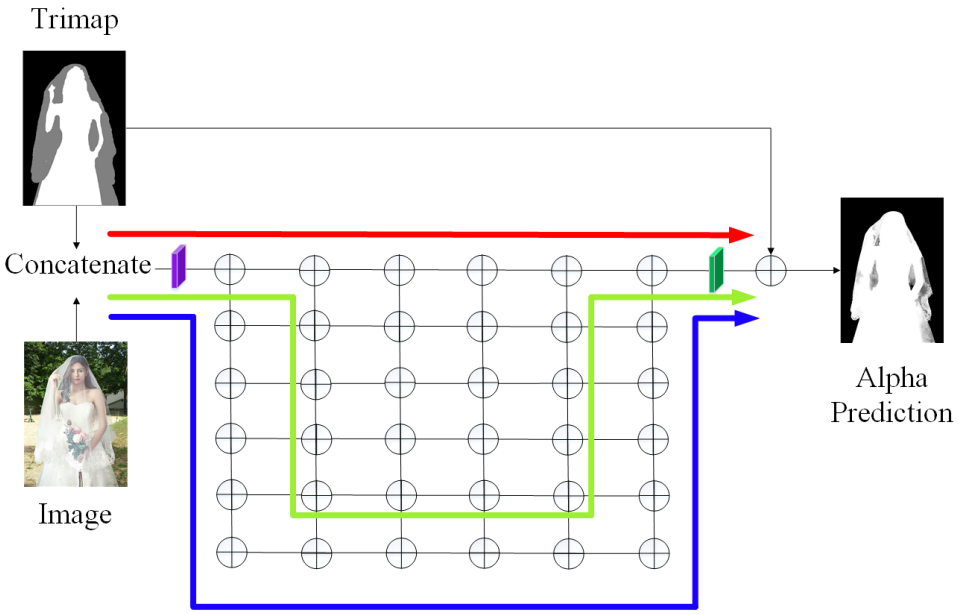


Figure 6.1: The information flow in our network.

Since we have proved that the Gridnet performs well for alpha matting task, we may try the same network structure for video matting task in the future work.

References

[1] Y. Aksoy, T. O. Aydin, and M. Pollefeys. Designing effective inter-pixel information flow for natural image matting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 228–236, July 2017.

[2] A. Al-Kabbany and E. Dubois. A novel framework for automatic trimap generation using the gestalt laws of grouping. In *Visual Information Processing and Communication VI*, volume 9410, page 94100G. International Society for Optics and Photonics, 2015.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994.

[4] L. Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of Computational Statistics*, pages 177–186. Physica-Verlag HD, Heidelberg, 2010.

[5] L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[6] Q. Chen, D. Li, and C. Tang. Knn matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(9):2175–2188, Sep. 2013.

[7] D. Cho, Y. Tai, and I. Kweon. Natural image matting using deep convolutional neural networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *European Conference on Computer Vision (ECCV)*, pages 626–643. Springer, 2016.

[8] Y.Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages II–II, Dec 2001.

[9] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units. In *The International Conference on Learning Representations (ICLR)*, 2016.

[10] C. Dong, C. C. Loy, K. He, and X. Tang. Image Super-Resolution Using Deep Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, Feb 2016.

[11] J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research (JMLR)*, 12(Jul):2121–2159, 2011.

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision (IJCV)*, 88(2):303–338, June 2010.

[13] D. Fourure, R. Emonet, E. Fromont, D. Muselet, A. Tremeau, and C. Wolf. Residual conv-deconv grid network for semantic segmentation. *arXiv preprint arXiv:1707.07958*, 2017.

[14] Eduardo S.L. Gastal and Manuel M. Oliveira. Shared sampling for real-time alpha matting. In *Computer Graphics Forum*, volume 29, pages 575–584. Wiley Online Library, 2010.

[15] R. Girshick. Fast r-cnn. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, Dec 2015.

[16] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, June 2014.

[17] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, 13–15 May 2010.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[19] L. Grady, T. Schiwietz, S. Aharon, and R. Westermann. Random walks for interactive alpha-matting. In *Visualization, Imaging and Image Processing (VIIP)*, pages 423–429, 2005.

[20] V. Gupta and S. Raman. Automatic trimap generation for image matting. In *International Conference on Signal and Information Processing (IConSIP)*, pages 1–5. IEEE, Oct 2016.

[21] K. He, C. Rhemann, C. Rother, X. Tang, and J. Sun. A global sampling method for alpha matting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2049–2056, June 2011.

[22] K. He, J. Sun, and X. Tang. Guided image filtering. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *European Conference on Computer Vision (ECCV)*, pages 1–14. Springer, 2010.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Dec 2015.

[24] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *European Conference on Computer Vision (ECCV)*, pages 630–645. Springer, 2016.

[26] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, July 2017.

[27] J.Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 379–387. Curran Associates, Inc., 2016.

[28] L. Karacan, A. Erdem, and E. Erdem. Image matting with kl-divergence based sparse sampling. In *IEEE International Conference on Computer Vision (ICCV)*, pages 424–432, Dec 2015.

[29] A. Karpathy and Lei F-F. A running demo of a CONV layer in the cs231n courses notes, March 2017. `http://cs231n.github.io/convolutional-networks/`.

[30] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *The International Conference on Learning Representations (ICLR)*, 2015.

[31] A. Krizhevsky, I. Sutskever, and G.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.

[32] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[33] P. Lee and Y. Wu. Nonlocal matting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2193–2200, June 2011.

[34] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):228–242, Feb 2008.

[35] C. Li, P. Wang, X. Zhu, and H. Pi. Three-layer graph framework with the sumd feature for alpha matting. *Computer Vision and Image Understanding*, 162:34–45, 2017.

[36] T-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[37] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, and A.C. Berg. Ssd: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *European Conference on Computer Vision (ECCV)*, pages 21–37. Springer, 2016.

[38] S. Lutz, K. Amplianitis, and A. Smolic. Alphagan: Generative adversarial networks for natural image matting. In *Proceedings of the British Machine Vision Conference (BMVC)*, page 259. BMVA Press, 2018.

[39] S. Mor-Yosef, D. Zeevi, A. Samueloff, M. Donhin, H. Frankfurter, and J.G. Schenker. Vaginal delivery following one previous cesarean birth: nation wide survey. *Asia-Oceania Journal of Obstetrics and Gynaecology*, 16(1):33–37, 1990.

[40] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814. Omnipress, 2010.

[41] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1520–1528, Dec 2015.

[42] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic Differentiation in PyTorch. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017.

[43] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[44] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016.

[45] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[46] C. Rhemann, C. Rother, and M. Gelautz. Improving color modeling for alpha matting. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 115.1–115.10. BMVA Press, 2008.

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

[48] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F-F. Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, Dec 2015.

[49] M. A. Ruzon and C. Tomasi. Alpha estimation in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 18–25, June 2000.

[50] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[51] E. Shahrian, B. Price, S. Cohen, and D. Rajan. Temporally coherent and spatially accurate video matting. In *Computer Graphics Forum*, volume 33, pages 381–390. Wiley Online Library, 2014.

[52] E. Shahrian and D. Rajan. Weighted color and texture sample selection for image matting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 718–725, June 2012.

[53] E. Shahrian, D. Rajan, B. Price, and S. Cohen. Improving image matting using comprehensive sampling sets. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 636–643, June 2013.

[54] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 39, pages 640–651, April 2017.

[55] X. Shen, X. Tao, H. Gao, C. Zhou, and J. Jia. Deep automatic portrait matting. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *European Conference on Computer Vision (ECCV)*, pages 92–107. Springer, 2016.

[56] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *The International Conference on Learning Representations (ICLR)*, 2015.

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

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

[59] T. Tijmen and H. Geoffrey. Lecture 6.5-RmsProp: Divide the Gradient by a Running Average of its Recent Magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[60] J. Wang and M. F. Cohen. An iterative optimization approach for unified image segmentation and matting. In *IEEE International Conference on Computer Vision (ICCV) Volume 1*, volume 2, pages 936–943, Oct 2005.

[61] J. Wang and M. F. Cohen. Optimized color sampling for robust matting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2007.

[62] J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 341–349. Curran Associates, Inc., 2012.

[63] L. Xu, J. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1790–1798. Curran Associates, Inc., 2014.

[64] L. Xu, J. Ren, Q. Yan, R. Liao, and J. Jia. Deep edge-aware filters. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, pages 1669–1678. PMLR, 2015.

[65] N. Xu, B. Price, S. Cohen, and T. Huang. Deep image matting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 311–320, July 2017.

[66] M.D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *European Conference on Computer Vision (ECCV)*, pages 818–833. Springer, 2014.

[67] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. Conditional random fields as recurrent neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1529–1537, Dec 2015.

[68] Y. Zheng and C. Kambhamettu. Learning based digital matting. In *IEEE International Conference on Computer Vision (ICCV)*, pages 889–896, Sep. 2009.