INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Enhanced Image Matting Algorithm using Local and Non-local Neighbors: A K-NN-based Color Modeling Approach

Ilias Ihadian

Bachelorarbeit

Date of issue:	03.05.2024
Date of submission:	05.08.2024
Reviewers:	Prof. Dr. Stefan Conrad
	Prof. Dr. T. Dickscheid

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 05.08.2024

Ilias Ihadian

Abstract

Alpha matting is a process of determining α values to determine the foreground and background precisely. The difficulty lies in determining the α values, especially in complex regions and fine details. Different algorithms were developed to solve this problem, such as the k-nearest-neighbor-matting (KNNM) or closed-form-matting (CFM). These algorithms have difficulties they need to overcome. The primary objective of this bachelor thesis is to develop a highly precise image matting algorithm. The proposed Local and Non-Local Color Line Model (LNCLM) achieves this by combining the strengths of existing models and overcoming their limitations. The LNCLM algorithm gets an image and a Trimap as inputs. Afterward, these inputs go through 3 stages. First, we search for the 2 different kinds of neighbors for each pixel. One kind of neighbor is the one we get through using knn, and the other is encompassed around the pixel in a window. Secondly, we calculate the 2 different Laplacian matrices for each of these to different neighbors. Moreover, lastly, we determine the α matte with these Laplacian matrices. The experimental results show that combining local and non-local principles gives us better results. We have better results compared to KNNM and CFM. Therefore, we can say that the proposed algorithm identifies holes, complex regions, and fine details better.

Contents

1 Introduction **1**
1.1 Motivation: background of the problem of image matting algorithms . . . 1
1.2 Goal: What does the mentioned algorithm try to do . . . 1

2 Foundations **2**
2.1 Trimap . . . 2
2.2 Local color line model . . . 3
2.3 K-Nearest-Neighbours . . . 3
2.4 Difference: Local Nonlocal Neighbors . . . 4

3 Related Work **4**
3.1 Closed-Form-Matting . . . 5
3.2 KNN-Matting . . . 7

4 Methods of Evaluation **8**
4.1 MSE . . . 8
4.2 SAD . . . 8
4.3 Dataset for evaluation . . . 9

5 LNCLM algorithm **9**
5.1 Introduction into the Nearest Neighbour color line model . . . 9
5.2 Feature Vector . . . 9
5.3 NN-Color-line Derivation . . . 10
5.4 Cost Function from NN-Color-line-model . . . 11
5.5 Combining Nearest Neighbors Color Line Model with Local Color Line Model . . . 12

6 Implementation **12**
6.1 Basic Implementation . . . 13
6.2 Finding the Neighbors . . . 13
6.3 Calculating the Laplacian Matrix . . . 14
6.4 Calculating the α matte . . . 16

7 Experimental Results **17**

8 Conclusion **19**

References **20**

List of Figures **21**

List of Tables **21**

1 Introduction

1.1 Motivation: background of the problem of image matting algorithms

The evolution and usage of Image-Matting algorithms have increased in popularity. The precise extraction of foreground objects is used in image and video editing. It also has several important uses in entertainment, advertising, and e-commerce. It is used for special effects where the foreground needs to be identified. Many e-commerce shops use it indirectly to create a clean look for their products by cutting out the background so that the focus remains on the product. image-matting algorithms allow us to separate foreground and background precisely.

The challenge in image-matting algorithms lies in finding a reliable and efficient method to determine how much each pixel belongs to the foreground or background. Traditional manual segmentation methods, for example, choosing the pixel with your brush tool on your own, while accurate, is time-consuming and not scalable for larger datasets. This is where image-matting comes in, aiming to simplify the process. However, these algorithms, like the ones that you will get introduced to later, are not without challenges. They struggle with accurately capturing fine details and textures, dealing with half-transparency, and managing complex scenarios with multiple objects.

Two well-known methods are K-Nearest-Neighbor-Matting (KNNM) and Closed-Form-Matting (CFM). K-Nearest-Neighbor-Matting [2] tries to use K-Nearest-Neighbor to identify the k pixels that are similar to each pixel. The publishers promise fast and precise foreground separation at holes and complex regions. Nevertheless, it tends to be less precise in smooth areas. Closed-Form-Matting [4] uses something that is called color line assumption (color line model). The publishers promise fast and precise foreground separation in smooth areas. Still, it has also been shown to be less precise at holes and complex regions.

In this bachelor thesis, we will examine the background and motivation behind Image-Mapping algorithms, also called Alpha-Matting algorithms. We will implement an algorithm, show its experimental results, and draw a conclusion based on its results.

1.2 Goal: What does the mentioned algorithm try to do

In this thesis, we aim to build an efficient method that solves the problems that both the KNNM and the CFM had before. This method should be able to separate precisely the foreground from the background in areas like holes, complex regions, and smooth areas.

Therefore, we will be implementing the algorithm [3] from the paper "Local and Nonlocal Color Line Models for Image Matting" which tries to combine the CFM algorithm with the KNNM algorithm. The algorithm will be called LNCLM algorithm.

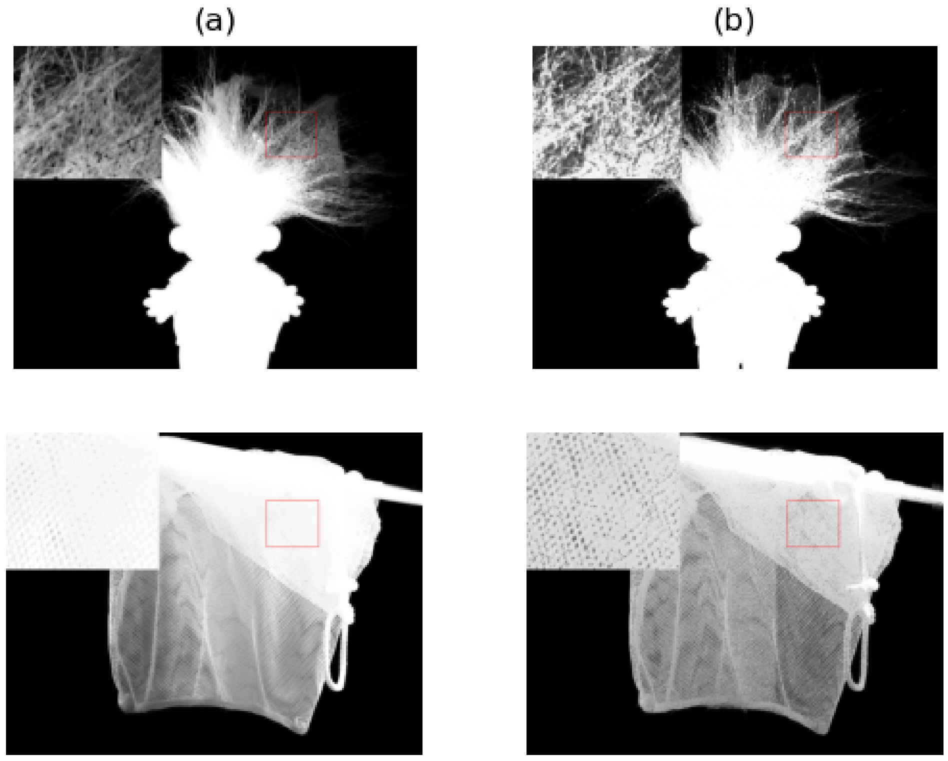

Figure 1: Difference between KNNM and CFM: (a) CFM and (b) KNNM.

2 Foundations

In this section, we will learn all the necessary methods to understand the algorithm from the paper "Local and Nonlocal Color Line Models for Image Matting". We will also understand how these foundations connect to the algorithm described.

2.1 Trimap

A Trimap is separated in 3 regions. The foreground, background and unknown. It is usually used in Alpha-matting algorithms and we will use it as a second input for the LNCLM algorithm. It usually created by the user of the algorithm. The foreground region is marked in white and covers the pixels that definitely belong to the foreground. The background region is marked in black and covers the pixels that definitely belong to the background. The unknown Region is marked in gray and covers the pixels where the user is unsure if it belongs to the background or foreground.

The Trimap's quality directly impacts the precision of the resulting segmentation. Subsequently, the Alpha-Matting algorithm takes the lead, employing various techniques, like

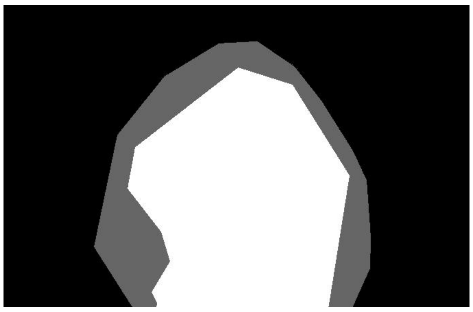

Figure 2: Example of a trimap

for example kNN or the local color line model, to discern the identity and extent of the unknown pixels, whether they belong to the foreground or background.

2.2 Local color line model

The Local color line model is a fundamental concept in the LNCLM algorithm. It posits that the pixels residing between the foreground and background of an image form a distinct color line in the RGB 3D space. To understand this further, we need to understand that the colors are not chosen randomly between the background and foreground. Furthermore, they tend to follow certain patterns and trends. These patterns can be visualized as color lines in color space. Therefore, we can write two similar formulas.

$$F_i = \beta_i^F F_1 + (1 - \beta_i^F) F_2 \tag{1}$$

$$B_i = \beta_i^B B_1 + (1 - \beta_i^B) B_2 \tag{2}$$

The variables F_1 and F_2 stand for two different pixel in the foreground and B_1 and B_2 stand for two different pixel in the background. If we imagine a straight line going from one to two, then β_i is defined as between $[0; 1]$ and stands for how near we go from pixel one to two. Here, we can see it visualized in Figure (3).

2.3 K-Nearest-Neighbours

The K-Nearest-Neighbours method is generally based on the concept of neighborhood, where the majority of its nearest neighbors determine the class assignment of an unknown observation. The parameter k determines the number of nearest neighbors used for classification. One crucial aspect of K-Nearest-Neighbours is that the algorithm learns from the existing training data and uses this information to classify new, unknown data. This flexibility makes the kNN classifier exceptionally versatile and adaptable to different types of data and problems [6]. This is the general use case of K-Nearest-Neighbours. However, the KNNM and LNCLM algorithms are used differently. Instead of classification, we only want to know the neighbors of a specific pixel. For example, if our k is 20 and the pixel of our choice is a random pixel, we want to know which 20 pixels are the

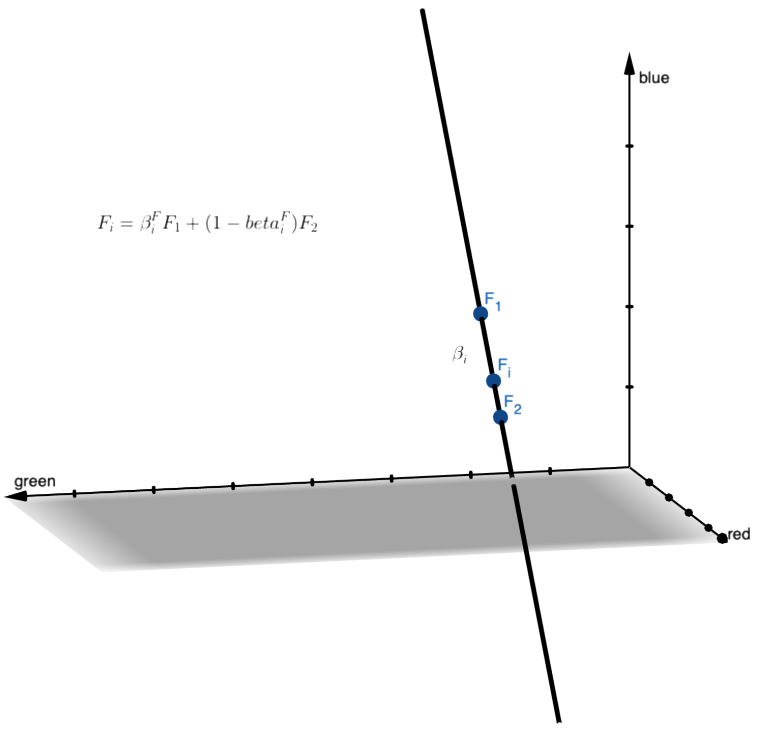

Figure 3: Pictorial example of the formula

closest to this random pixel. Nearest in the context of Image-Matting means a similar appearance and spatial nearest [3].

2.4 Difference: Local Nonlocal Neighbors

The difference between Local and Nonlocal Neighbors lies in the selection of the neighbors. The local neighbors are the pixels that sit around a specific pixel in a window (**??**). This approach is used in CFM. The Nonlocal Neighbors are pixels that look similar to the pixel we are looking at (4). This approach is used in KNN-Matting. These neighbors should share the same alpha-values as the pixels looked at. Both approaches are later used in the LNCLM algorithm. [3]

3 Related Work

In this section, we will learn the related works that will help us further comprehend the LNCLM algorithm. We will also understand how these related works connect to the described algorithm (section: LNCLM algorithm).

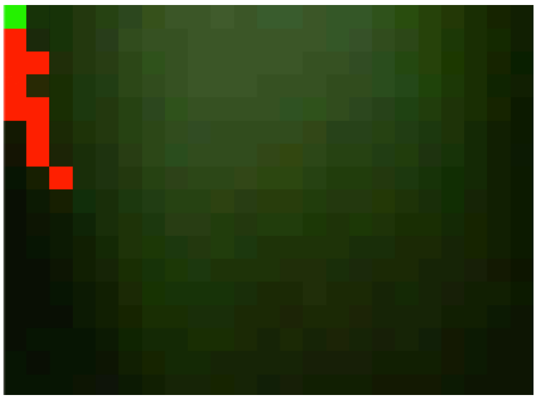

Figure 4: Example for KNN

If we choose k = 10 and want to search for the k-Nearest-Neighbor for the green pixel, then the red ones are the neighbors

3.1 Closed-Form-Matting

CFM aims to find the optimal alpha matte by minimizing a cost function that shows the difference between the estimated alpha value and the ground truth alpha value of all pixels in the image based on the color line model. Let us go through the steps together to understand the derivation of the cost function in closed-form matting. First of all, we will learn about the matting equation:

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i. \tag{3}$$

The variable I_i is the color value of the ith pixel while F_i and B_i are the foreground and background color values of the ith pixel. The variable α_i has the range [0,1] and stands for the foreground opacity [4]. This equation is the basis of the matting problem, where the goal is to get a near approximation of the absolute α.

The color line model(subsection: Local color line model) assumes that the pixels of the background and foreground over a small window are approximately constant. Because of this assumption, we can build this equation:

$$\alpha_i \approx a I_i + b. \tag{4}$$

While a is the gradient or slope of this linear function, and it is defined in [4] as

$$a = \frac{1}{F - B} \tag{5}$$

and b is seen as the y-intercept and is defined in [4] as

$$b = -\frac{B}{F - B}. \tag{6}$$

Now, for the first time, we will define the cost function. We will start by defining it over a window w_j around each pixel j:

$$J(\alpha, a, b) = \sum_j \sum_{i \in w_j} (\alpha_i - a_j I_i - b_j)^2 + \epsilon a_j^2. \tag{7}$$

The second term ϵa_j^2 is an error term to ensure numerical stability. The goal is that $\alpha_i - a_j I_i - b_j)^2$ this term will be zero for every pixel and this is how the equation is going to be reformulated:

$$(a_j, b_j) = \text{argmin}_{a,b} \sum_{i \in w_j} (\alpha_i - aI_i - b)^2 + \epsilon a^2 \tag{8}$$

By solving this minimization problem, we can reformulate the function again:

$$J(\alpha, a, b) = \sum_k ||G_k \begin{pmatrix} a_k \\ b_k \end{pmatrix} - \alpha_k||^2. \tag{9}$$

The variable G_k is a matrix that contains the color information of the window k, and α_k is a vector of αvalues. The optimal a and b are now given by

$$\begin{pmatrix} a_k \\ b_k \end{pmatrix} = \left(G_k^T G_k\right)^{-1} G_k^T \alpha_k \tag{10}$$

Substituting the optimal a and b back into the cost function, we get this form:

$$J(\alpha) = \sum_k \bar{\alpha}_k^T \bar{G}_k^T \bar{G}_k \bar{\alpha}_k \tag{11}$$

The variable $\bar{G}_k$ is defined as $I - G_k(G_k^T G_k)^{-1} G_k^T$ Moreover, $\bar{G}_k^T \bar{G}_k$, which will later be the Laplacian matrix L, is an $N \times N$ matrix whose (i,j)-th entry is:

$$L_{ij} = \sum_{k|(i,j) \in w_k} \left(\delta_{ij} - \frac{1}{|w_k|}(1 + (I_i - \mu_k)(\sigma_k + \frac{\epsilon}{|w_k|} I_3)^{-1}(I_j - \mu_k)) \right) \tag{12}$$

The variable δ_{ij} is the Kronecker delta, σ_k and I_3 are 3×3 covariance and identity matrices, and μ_k is a 3×1 mean vector of the colors in the k-th window.

After we changed the cost function, we extended the linear model 4 to include all color channels separately:

$$\alpha_i \approx \sum_c a_c I_{ic} + b. \tag{13}$$

When we now fuse 12 and 13. Therefore, we get this quadratic form:

$$J(\alpha) = \alpha^T L \alpha \tag{14}$$

Finally, to find the optimal alpha matte, we equate the cost function to 0 like this:

$$0 = L\alpha \tag{15}$$

Compared to LNCLM, CFM only uses local neighbors, while LNCLM also uses non-local neighbors. The CFM algorithm searches consider the pixels around each pixel, which further leads to smooth segmentations where there should be harsh ones. The LNCLM uses the k nearest pixels and leads to more precise α-values in holes and complex areas.

3.2 KNN-Matting

The Goal of KNNM is to find the optimal alpha matte by minimizing a cost function that shows the difference between the estimated alpha value and the ground-truth alpha value of all pixels in the image based on k nearest neighbours. Let us go through the steps together to understand the derivation of the cost function in KNNM.

First of all, the Laplacian matrix is defined [2]:

$$L = D - A. \tag{16}$$

The variable D is an N × N diagonal matrix, where N is the total number of pixels, and A is the nonlocal affinities matrix. The nonlocal affinities matrix shows us how [2] is defined as:

$$A = [k(i,j)] \tag{17}$$
$$k(i,j) = \exp(-\frac{1}{h_1^2}||X(i) - X(j)||_g^2 - \frac{1}{h_2^2}d_{ij}^2). \tag{18}$$

For better understanding, we choose pixels i and j are the pixels most similar to those chosen by k nearest neighbor. The feature vectors X(i) and X(j) are computed at pixel i or j. The constants h_1 and h_2 are found by trying, d_ij is the distance between the pixels i and j, and $|| \cdot ||_g$ is a norm weighted by a center-weighted Gaussian.

Now the cost function gets introduced [2]:

$$g(x) = x^T Lx + \lambda \sum_{i \in m-v} x_i^2 + \lambda \sum_{i \in v} (x_i - 1)^2 \tag{19}$$

Just to clarify J, from (subsection: Closed-Form-Matting), and g stand both for cost function and x stands for α. The term $\lambda \sum_{i \in m-v} x_i^2$ handles the unmarked pixels and $\lambda \sum_{i \in v} (x_i - 1)^2$ ensures that marked pixels are close to their specified values.

The cost function gets reformulated and put into the matrix form:

$$g(x) = x^T Lx + \lambda \sum_{i \in m} x_i^2 - 2\lambda v^T x + \lambda|v| \tag{20}$$
$$= \frac{1}{2}x^T 2(L + \lambda D)x - 2\lambda v^T x + \lambda|v| \tag{21}$$

The vector v is a binary vector for the marked pixels.

Through deriving and equating the cost function with zero, we get the optimal alpha matte:

$$x = (L + \lambda D)^{-1}(\lambda v) \tag{22}$$

Compared to LNCLM, KNNM only uses non-local neighbors, while LNCLM also uses local neighbors. The KNNM algorithm searches for the k nearest pixels for each pixel based on color and spatial information, which does not include any smoothness assumption, which further leads to harsher segmentations. This is compared to LNCLM, which also uses a local window around each pixel and therefore, leads to more precise α-values in smooth and complex areas.

4 Methods of Evaluation

In this section, we will learn some evaluation methods to evaluate the algorithm from the paper "Local and Nonlocal Color Line Models for Image Matting."

4.1 MSE

The mean squared error is a risk function that uses quadratic loss [1, p. 20]. It calculates the squared differences between observed and estimated values. The formula for MSE is

$$\mathrm{MSE}(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \theta)^2]. \tag{23}$$

Where $\hat{\theta}$ is the estimator of θ. For our practical calculations, we use a transformation of the formula [3]:

$$\mathrm{MSE}(\alpha^*) = \left(\frac{\sum_i (\alpha_i - \alpha_i^*)^2}{N_{\text{unk}}} \right). \tag{24}$$

Where α^* are the predictions of the α values. The variable N_{unk} is the number of pixels in the unknown region of the Trimap. In this derivation of the function, we go through all the α values and subtract the predicted alpha value from our true alpha value, also called ground truth, which then gets squared and divided by the number of unknown pixels. A low MSE indicates better performance because the model's predictions are highly accurate and closely match the actual observed values. A high MSE suggests bad performance because the model's predictions are not accurate and does not closely match the actual observed values. The MSE helps us compare the different alpha matting methods and identify the one with the most accurate alpha value estimations. It is also one of two evaluation metrics in the paper [3] to compare the LNCLM algorithm.

4.2 SAD

The sum of all differences is another evaluation metric. It calculates the differences between observed and estimated values. The formula for SAD is

$$\mathrm{SAD} = \sum_{i=1}^{m} \sum_{j=1}^{n} |A(i,j) - B(i,j)|. \tag{25}$$

The variables $A(i,j)$ and $B(i,j)$ are the pixels at position (i,j) in image A and B. The constants m and n are the height and width of these images. For our practical calculations, we use a derivation of the formula [3]:

$$\mathrm{SAD}(\alpha^*) = \sum_i |\alpha_i - \alpha_i^*|. \tag{26}$$

Where α^* are the predictions of the α values. In this derivation of the function, we go through all the α values and subtract the predicted alpha value from our true alpha value.

A low SAD indicates better performance and a high SAD suggests bad performance. The SAD also helps us compare the different alpha matting methods and identify the one with the most accurate alpha value estimations. It is the second evaluation metric in the paper [3] to compare the LNCLM algorithm.

4.3 Dataset for evaluation

We used alphamatting.com as a dataset to evaluate our code. It was first proposed in a paper [5] and was meant to be the first standard library for image matting algorithms to be tested. Already existing datasets to that point were affected by noise, which refers to random pixels that do not represent the actual scene. Also, they were primarily biased towards specific algorithms. One had, for example, more smooth images, which would favor algorithms like CFM, and others had more images with hard segmentations, which would favor algorithms like KNNM. Most old datasets had high-quality images, though they included artificial backgrounds and lacked diversity. The improved dataset [5] is augmented with natural scenes, diverse focus settings, and translucent objects. It provides data, scripts for community contributions, and enables interactive analysis of new results. Most importantly, the new dataset reveals the weaknesses and strengths of matting algorithms so we can reliably compare our LNCLM algorithm with others.

5 LNCLM algorithm

In this section, we will learn the algorithm proposed in the paper "Local and Nonlocal Color Line Models for Image Matting" [3]. It segments the foreground from the background, and we will especially have a look at the Nearest Neighbour color line model.

5.1 Introduction into the Nearest Neighbour color line model

As already mentioned in (section: Motivation), the CFM tends to be very precise in smooth areas, for example, with fur, but unprecise in complex areas, for example, with nets and glass. Meanwhile, the KNNM tends to be the complete opposite. It is exact in complex areas and very imprecise in smooth areas. That is why the authors wanted a color line model that combined the benefits of both Matting algorithms. This resulted in the Nearest Neighbour color line model. The premise of the Nearest Neighbour color line model, short NNCLM, is that the nearest neighbors of the feature vectors follow the color line model. The feature vector will be explained in the following subsection.

5.2 Feature Vector

The feature vectors represent each pixel in multidimensional space and include spatial and color information. This helps to find the nearest neighbor to each pixel. The feature vector proposed in the paper [3] is

$$X(i) = (I^r, I^g, I^b, x, y)_i. \tag{27}$$

The variable i stands for the pixel i. The values I^r, I^g, and I^b stand for the RGB color information that the pixel I contains. While x and y represent spatial information. In our implementation of LNCLM, we weight the spatial coordinates with dws to make them less important.

```
1: $f = [I_r, I_g, I_b, \text{dws} \cdot x, \text{dws} \cdot y]$
```

The distance between two pixels is called the Euclidean distance and is calculated like this:

$$\text{Euclidean distance} = ||X(i) - X(j)||. \tag{28}$$

This is used to find the nearest neighbors through K-Nearest-Neighbor.

5.3 NN-Color-line Derivation

The assumption is that the NNCLM and the KNN follow the color-line model. Figure 6 shows that the assumption is valid for smooth and complex regions. You can see the linear distribution of KNN foreground and background.

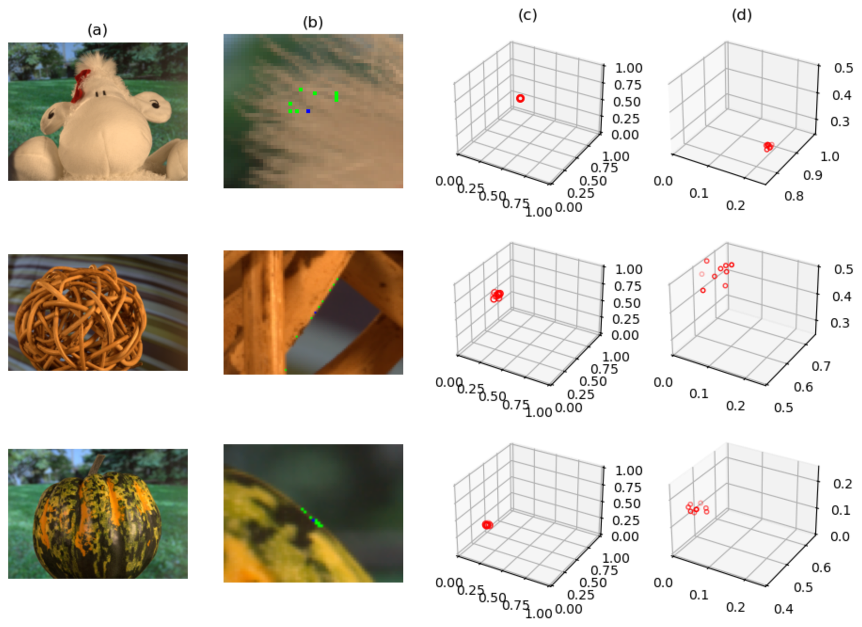

Figure 5: Examples showing that KNN satisfies the color line model (K = 9). (a) original images, (b) zoomed-in images. The green pixel (marked) are the KNN of the blue pixel (also marked). (c) foreground and background color distributions of the KNN in (b), respectively. (d) zooms into (c).

However, this does not mean that the feature vectors are linearly aligned, but the color components are. Because of that, the foreground and background can be represented by convex functions(1) and (2). The value N_i represents the neighbor vectors for a specific pixel i. The variables F_1, F_2, B_1, and B_2 are constant colors over N_i (**??**) . As we already explained in (section: Local color line model). This formula and the matting equation used in CFM (3) can be combined. Therefore, the pixel for all j in N_i can be described as

$$I_j = \alpha_j(\beta_j^F F_1 + (1-\beta_j^F)F_2) + (1-\alpha_j)(\beta_j^B B_1 + (1-\beta_j^B)B_2), \forall j \in N_i. \tag{29}$$

We can obtain a 4D model out of this formula, as in CFM

$$\alpha_j = a_i^R I_j^R + a_i^G I_j^G + a_i^B I_j^B + b_i, \forall j \in N_i. \tag{30}$$

The variables I_j^R, I_j^G, and I_j^B are the red, green, and blue values of the jth pixel. While a_i^R, a_i^G, and a_i^B are the gradients or slope of this polynomial function and they are defined in [4] as

$$a = \frac{1}{F-B}. \tag{31}$$

and b is seen as the y-intercept and is defined in [4] as

$$b = -\frac{B}{F-B}. \tag{32}$$

5.4 Cost Function from NN-Color-line-model

The 4d model (30) can be denoted to this cost function

$$J_n = \sum_{i \in I} \left(\sum_{j \in N_i} \left(a_j - \sum_C a_i^C I_j^C - b_i \right)^2 + \epsilon \sum_C \left(a_i^C\right)^2 \right). \tag{33}$$

The variable I^C is the entire image, C is the different color channels, and ϵ is a constant for regularization. The cost function has $5N$ unknowns and α, a_i^C, and b_i are quadratic. The variables a_i^C and b_i can be eliminated by Theorem 2 in [4]. Which reduces the cost function to only depend on α. Now, the formula can be rewritten into

$$J_n = \alpha^T L_n \alpha. \tag{34}$$

The variable α is an $N \times 1$ vector, and L_n is an $N \times N$ Laplacian matrix. The (j,k)th element of the Laplacian matrix L_n is

$$\sum_{i|(j,k) \in N_i} \left(\delta_{jk} - \frac{1}{|N_i|} \left(1 + (I_j - \mu_i)^T \left(\Sigma_i + \frac{\epsilon}{|N_i|} I_3 \right)^{-1} (I_k - \mu_i) \right) \right). \tag{35}$$

The variable δ_{jk} is the Kroenecker delta, which is one if j is equal to k else it is 0. The variable Σ_i is a three-by-three covariance matrix, which contains the covariances of the three color channels, μ_i is the mean vector of the three color channels, and I_3 is a three by three Identity matrix. The intuition behind the simplification is that by eliminating a_i^C and b_i, the cost function J_n becomes a quadratic form in terms of a. This simplification minimizes the quadratic form and using linear algebra methods we can find the alpha values. The Laplacian matrix L_n contains the relationship between each pixel and each other, considering both color information and spatial relationships within neighborhoods N_i.

Because of this cost function, we now have several advantages. Due to the nonlocal principle, the LNCLM is effective across different images with color variations. Also, by considering all color variations, the LNCLM can reflect foreground and background variations more accurately, leading to more accurate alpha mattes. Therefore, the LNCLM performs better in images with a lot of color variation, for example, with fur. It overcomes the limitations of previous methods by not solely relying on appearance similarity. Nevertheless, it still needs improvement.

5.5 Combining Nearest Neighbors Color Line Model with Local Color Line Model

The authors now combine the cost functions of the Nearest-Neighbors color line model and the local color line model

$$\begin{aligned} J &= \mu J_n + (1-\mu)J_l + \lambda(\alpha^T - b_S^T)D_S(\alpha - b_S) && (36)\\ &= \mu\alpha^T L_n\alpha + (1-\mu)\alpha^T L_l\alpha + \lambda(\alpha^T - b_S^T)D_S(\alpha - b_S). && (37)\end{aligned}$$

The constant μ balances the cost functions between the local and NN color line models. While λ is a large number for the smoothness term. The diagonal matrix D_S indicates a constrained pixels. Also, the vector b_S contains specified alpha values for the constrained pixels.

The reason for that is that NNCLM is not as smooth as anticipated. This leads us also to apply the local color line model to the image. Now, we have the results of the NNCLM and the local color line model, which can now be fused. Only applying one will lead us to either too segmented rough of an α matte or too smooth.

By calculating the maximum of J, through differentiating the cost function and setting the derivatives to zero, we get this function:

$$\alpha = (\mu L_n + (1-\mu)L_l + \lambda D_S)^{-1}\lambda b_S. \tag{38}$$

6 Implementation

With our newly acquired theoretical knowledge, we are now ready to apply the LNCLM algorithm in practical scenarios.

6.1 Basic Implementation

This subsection will provide a comprehensive overview and a detailed understanding of the LNCLM algorithm structure, equipping us with in-depth knowledge that will be further expanded in the subsequent sections. Like most alpha matting algorithms, the LNCLM algorithm gets an image and a Trimap as input. These inputs are used to calculate the cost function, resulting in us getting our alpha matte. This alpha matte can then be applied via the matting equation 3 to get our foreground and background. The now shown pseudo code will clarify this further:

Algorithm 1 LNCLM Algorithm

```
1: **Input:** image and trimap
2: **Output:** alpha matte

3: neighborsnnclm ← the neighbors for each pixel calculated by kNN
4: neighborscf ← the neighbors for each pixel in a window

5: $L_n$ ← calculated Laplacian matrix with neighborsnnclm as input
6: $L_l$ ← calculated Laplacian matrix with neighborscf as input

7: $\alpha$ ← calculated alpha matte with $L_n$ and $L_l$ as input
8: **return** $\alpha$
```

The variable neighborsnnclm contains the neighbors to every pixel in the image. They are calculated by kNN, similarly to neighborscf, which contains the neighbors to every pixel. These neighbors are simply just the pixels that are in the window area of the chosen pixel. Both L_n and L_l are both Laplacian matrices, which are calculated with the neighbors as input and are calculated by this formula 35. The α-value gets L_n as well as L_l as input and solves this equation (38).

6.2 Finding the Neighbors

In this subsection, we will learn how to find the neighbors for the NNCLM and CFM parts of the LNCLM Algorithm.

The explanation is based on this pseudo-code:

As input we get the image, dws, and nn. The variable dws is weights the spatial coordinates of each pixel. The variable nn lets us decide how many neighbors we want to get for each pixel. In [3] dws is 0.5 and nn is 9. In lines 3 and 4, we get the height, width, and RGB color information of the image. Afterward, we build a coordinate system so that we can associate every pixel with a spatial coordinate. The values r, g, b, x, and y now get passed into the feature vector, which we explained beforehand in (subsection: Feature Vector). The feature vector now gets used by knn, which lets us find our nearest neighbors based on the feature vector. We use FLANN like in [3], an easy and fast way to implement kNN. These neighbors now get returned.

Now to searching neighborscf:

Algorithm 2 Searching for neighborsnnclm

```
1: Input: image, dws, nn
2: Output: neighborsnnclm

3: h, w ← image.shape
4: r, g, b ← image.colors

5: x, y ← Coordinates for every pixel

6: f ← $[r, g, b, \text{dws} \cdot x, \text{dws} \cdot y]$
7: neighborsnnclm ← knn(f, nn)
   return neighborsnnlcm
```

Algorithm 3 Searching for neighborscf

```
1: Input: image, radius
2: Output: neighborscf

3: h, w ← image.shape

4: for i = radius to h - radius do
5:     for j = radius to w - radius do
6:         Add the pixels in the window around pixel ij to neighborscf
   return neighborscf
```

The radius is the radius of the window around pixel ij. In [3], it is 1. Finding the neighbors for neighborscf is as easy as explained before. We go through each pixel of an image and put the neighboring pixels around the pixel ij into neighborscf. The only obstacle is the pixel at the edge of the image because they can not have a window of neighbors without exceeding the edge of the image. That is why we exclude them in the loops.

To make this part faster, we used njit from numba. A compiler speeds up our code by generating optimized machine code. We also used numba.prange to parallelize the nested for loops.

6.3 Calculating the Laplacian Matrix

The Laplacian matrix is calculated using the formula (35). The code shown now works out the formula in practice.

The Inputs are the image, trimap, and either neighborscf or neighborsnnclm because this method gets used twice, once with neighborscf and another time with neighborsnnclm. We initialize the arrays i, j, v, and load all known pixels from the trimap. Which are the pixels outside of the unknown region. Afterward, we start a for loop in which we go through every pixel in the image. In every iteration, we load the neighbors of the now chosen pixel. If these pixels are outside of the unknown region, so known then we skip

Algorithm 4 Calculating the Laplacian matrix

```
1: Input: image, trimap, neighbors
2: Output: Laplacian matrix L
3: Initialize $i$, $j$, and $v$ as arrays
4: known ← load all known pixels from trimap
5: for pixels ∈ image do
6:     nn ← load neighbors for this pixels

7:     if nn are all known then
8:         continue

9:     for dc in range(3) do
10:        $\mu_i$ ← mean of nn in every color channel

11:    $c$ ← nn $-$ $\mu$
12:    cov ← covariance matrix of $c$
13:    covtmp ← cov + ($\epsilon$/ nn.length ) * $I_3$
14:    inv ← invert covtmp

15:    for di ∈ nn do
16:        for dj ∈ nn do
17:            Add spatial coordinates of di to $i$
18:            Add spatial coordinates of dj to $j$

19:            tmp ← c[di].dot(inv).dot(c[dj])

20:            if i == j then
21:                tmp ← 1.0 - (1 + tmp) / nn.length
22:            else tmp ← 0.0 - (1 + tmp) / nn.length
23:            Add tmp to $v$
24: $L$ ← Build matrix L with the values $v$ at the coordinates $(i, j)$ return $L$
```

this iteration because α value is either 0 or 1. We now calculate the μ_i of every neighbor in every color channel. We now calculate in 35 the terms $I_j - \mu_i$ and $I_k - \mu_i$. However, we also calculated this for every neighbor so that we could use it later. To calculate the term $(\sigma_i + \frac{\epsilon}{|N_i|}I_3)^{-1}$ we compute the covariance of c , add to it $\frac{\epsilon}{nn.length}I_3$, and invert it completely. Finally, we started to go through the neighbors with di and dj. In every iteration of this nested loop we get their spatial coordinates, add them to either i or j, calculate the term $(I_j - \mu_i)^T(\sigma_i + \frac{\epsilon}{|N_i|}I_3)^{-1}(I_k - \mu_i)$ which we save in tmp. Furthermore, we see if i and j are equal, we add a one to tmp. If not, we do not. This gets added to v and we finally build our Laplacian matrix L, which we return.

We successfully calculated the formula (35) and got our Laplacian matrix. To make the code faster, we again used, like in (subsection: Finding the Neighbors), njit and prange from numba.

Algorithm 5 Calculating the α matte

```
1: Input: image, trimap, $L_l, L_n$
2: Output: $\alpha$ matte
3: known $\leftarrow$ load all known pixels from trimap
4: $D \leftarrow$ put every known pixel into a diagonal matrix

5: $\lambda \leftarrow 100.0$
6: $\mu \leftarrow 0.5$

7: $A \leftarrow \lambda * D + \mu * L_n + (1 - \mu) * L_l$

8: $b \leftarrow \lambda * known$

9: $\alpha \leftarrow$ solve$(A, b)$
10: $\alpha \leftarrow$ normalize $\alpha$ between 1 and 0
    return $\alpha$
```

6.4 Calculating the α matte

We are in the final stage of our LNCLM algorithm, in which we finally get the α matte.

We get the image and the trimap as input. We load the known pixels from the trimap as we did in (subsection: Calculating the Laplacian Matrix). Out of this, we build D by putting the information into a diagonal matrix. Its diagonal elements contain either 1 for all the constraint pixels and 0 otherwise. In line 5 and 6 we initialize λ and μ with 100.0 and 0.5 exactly like in [3]. Now we calculate the equation 38 separately. At first we determine the term $\mu L_n + (1 - \mu)L_l + \lambda D_S$ and save it in A. Secondly, we compute the term λb_S and save it in b. We can reformulate 38 like this:

$$\alpha = (\mu L_n + (1 - \mu)L_l + \lambda D_S)^{-1} \lambda b_S \tag{39}$$

$$\alpha = A^{-1} b \tag{40}$$

$$A\alpha = b. \tag{41}$$

$$\tag{42}$$

The term $A\alpha = b$ can now be solved by various methods, which we are getting into later. This α now gets normalized in the range of 0 and 1. This is returned, and we finally get our α matte.

To make the code faster, we used 2 things. First, we used a preconditioner, more specifically, the smoothed_aggregation_solver as a precondition from pyamg. It preconditions the matrix into a more suitable form for solving a matrix equation. Secondly, we used a conjugate gradient solver to solve the matrix equation $A\alpha = b$. It helps us solve the equation faster.

7 Experimental Results

In this section, we compare the LNCLM algorithm to the CFM and KNNM algorithms and see if we can build an efficient method that solves the problems that both the KNNM and the CFM had before. We compare the algorithm based on 27 training images with ground truth and 8 images without ground truth from [5]. We let CFM, KNNM, and LNCLM calculate α-mattes for all of these images, and we compare them to the ground truth with MSE and SAD. Like in [3] we set $K = 9, \epsilon = 10^{-7}, \mu = 0.5,$ and $= 100.$

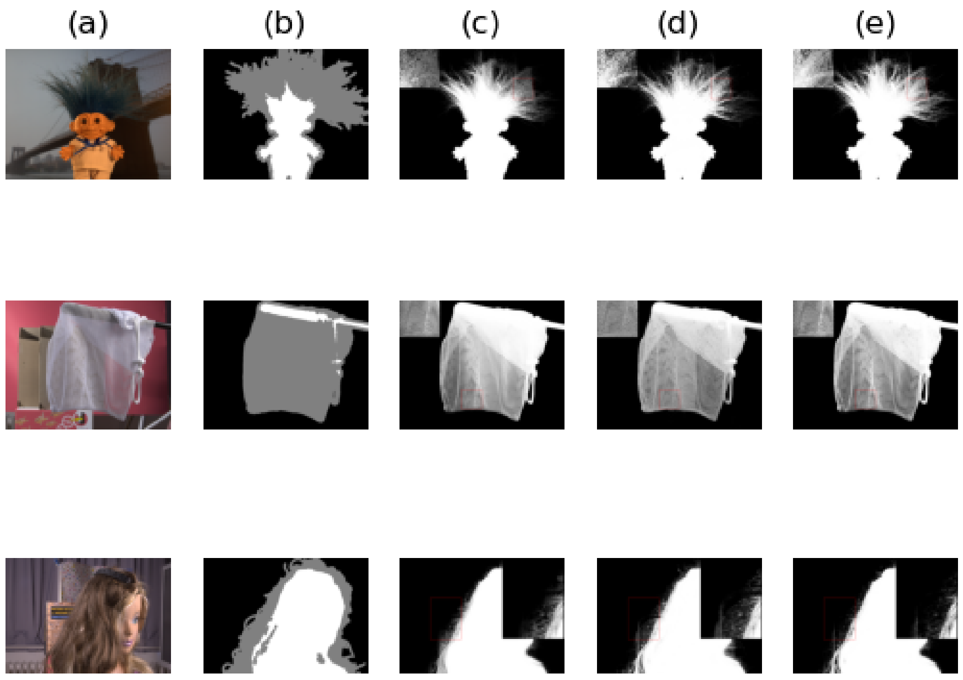

Figure 6: Matting results for test images: (a) original images, (b) Trimaps (c) CFM algorithm, (d) KNNM algorithm, (e) LNCLM algorithm.

In Figure 6, we can see that KNNM has harder segmentations and LNCLM is closer to the ground truth because of the combined local and NN color line models and the lack of smoothness assumption in KNNM. We also see that LNCLM better determines holes and complex regions due to its non local principle[3].

A quantitative comparison 1 shows that LNCLM outperformance KNNM and CFM in SAD and MSE. The reason is that LNCLM combines the strengths of both CFM and KNNM, giving us α-mattes that cancel out each weakness.

However, if we compare our MSE and SAD values to the values portrayed in table 2

	CFM	KNN	NNCLM	LNCLM
MSE	$2.18 * 10^{-2}$	$2.44 * 10^{-2}$	$2.12 * 10^{-2}$	$1.81 * 10^{-2}$
SAD	$5.32 * 10^{3}$	$6.11 * 10^{3}$	$5.22 * 10^{3}$	$4.78 * 10^{3}$

Table 1: MSE and SAD of our matting results.

	CFM	KNN	NNCLM	LNCLM
MSE	$2.18 * 10^{-2}$	$2.44 * 10^{-2}$	$1.97 * 10^{-2}$	$1.75 * 10^{-2}$
SAD	$5.32 * 10^{3}$	$6.11 * 10^{3}$	$5.00 * 10^{3}$	$4.75 * 10^{3}$

Table 2: MSE and SAD of the matting results from the paper.

by the paper [3] and also compared to the alpha mattes shown on [5], we see that our implementation is not as good as theirs. Figure 8 and table 2 shows the differences.

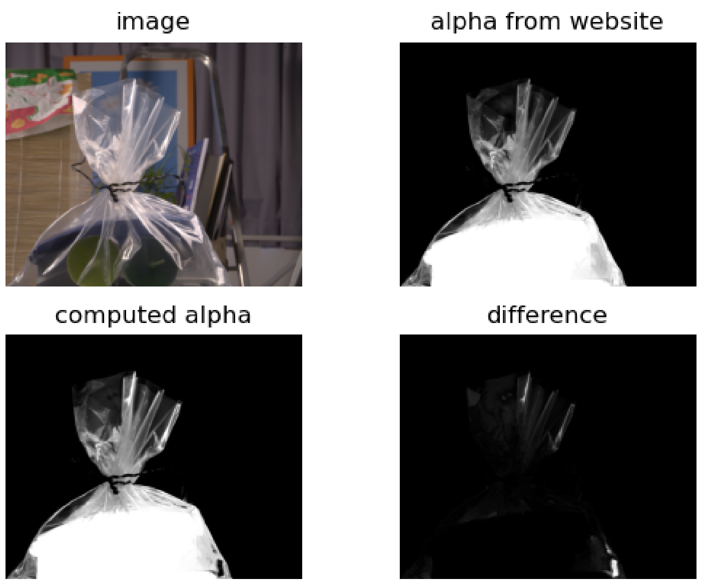

Figure 7: Comparison between our implementation and the paper implementation

We see that the CFM gives us the same MSE as well as SAD but a different MSE and SAD when it comes to the NNCLM, which means that the problem lies in the NNCLM. Furthermore, because we use the Algorithm part (4) to calculate the Laplacian matrix both with neighbors of neighborscf and neighborsnnclm, it can not contain the problem because our values for CFM are correct. Therefore, the problem needs to lay in 2. To solve this issue, we tried to change the parameters. First we tackled dws from the feature vector. We made the spatial coordinates more and less critical and concluded that a dws of 0.6 gave us the α mattes nearest to the paper [3]. Secondly, we tried changing K in an attempt to place a more significant or lesser emphasis on the nonlocal principle. After testing different values for K, we concluded that the preconditioned K is already the best. Last but not least, we also tried using KNNM instead of NNCLM for no particular reason, which did not help.

8 Conclusion

We introduced and evaluated an enhanced image matting algorithm using local and non-local neighbors. Our suggested algorithm, LNCLM, overcomes the drawbacks of earlier techniques such as KNN-Matting (KNNM) and Closed-Form-Matting (CFM) by combining the advantages of the local color line model and the Nearest-Neighbors color line model. We aimed to overcome problems like over-smoothing hole regions, correctly identifying complex regions, and being more precise in foreground segmentation. We conducted on 27 training images with ground truth and 8 images without ground truth and we found out that LNCLM outperforms KNNM and CFM both in MSE and SAD. We improved performance due to combining the strengths of local and non-local approaches. The current implementation shows promising results and future work could focus on refining the parameters and enhancing the computational efficiency. We aim for performance comparable to state-of-the-art implementations.

References

[1] P.J. Bickel and K.A. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics*. Second Edition. Mathematical Statistics: Basic Ideas and Selected Topics v. 1. Prentice Hall, 2001.

[2] Qifeng Chen, Dingzeyu Li, and Chi-Keung Tang. "KNN matting". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35.9 (2013), pp. 2175–2188.

[3] Byoung-Kwang Kim, Meiguang Jin, and Woo-Jin Song. "Local and nonlocal color line models for image matting". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 97.8 (2014), pp. 1814–1819.

[4] Anat Levin, Dani Lischinski, and Yair Weiss. "A Closed-Form Solution to Natural Image Matting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (2008), pp. 228–242.

[5] Christoph Rhemann, Carsten Rother, Jue Wang, Margrit Gelautz, Pushmeet Kohli, and Pamela Rott. "A perceptually motivated online benchmark for image matting". In: June 2009, pp. 1826–1833.

[6] Zhongheng Zhang. "Introduction to machine learning: K-nearest neighbors". In: *Annals of Translational Medicine* 4.11 (June 2016).

List of Figures

1 Difference between KNNM and CFM: (a) CFM and (b) KNNM. 2
2 Example of a trimap 3
3 Pictorial example of the formula 4
4 Example for KNN 5
5 Examples showing that KNN satisfies the color line model (K = 9). (a) original images, (b) zoomed-in images. The green pixel (marked) are the KNN of the blue pixel (also marked). (c) foreground and background color distributions of the KNN in (b), respectively. (d) zooms into (c). 10
6 Matting results for test images: (a) original images, (b) Trimaps (c) CFM algorithm, (d) KNNM algorithm, (e) LNCLM algorithm. 17
7 Comparison between our implementation and the paper implementation 18

List of Tables

1 MSE and SAD of our matting results. 18
2 MSE and SAD of the matting results from the paper. 18