目录

1 应用领域背景

在当下快速发展的数字时代，数字图像充斥在生产生活的方方面面，更逐步成为人们获取外界信息的主要媒介。相应的图像处理技术已然成为当今计算机科学领域的一个重要的研究方向，并且已经在医学图像处理、机器视觉、目标追踪等方向有了广泛的应用。

图像抠图(Image Matting)技术更是数字图像处理技术中的重要一环，人们对于抠图技术的需求也超过了从前的任何一个时代，从个人的修图到影视创作都需要抠图技术的参与。比如在电视电影制作中，使用抠图技术可以将演员“抠出”并融合到新的场景中，结合特效技术，可以制作出美轮美奂、亦真亦假的特效场景。并且普通人对抠图技术也有着巨大的需求，从对拍摄的照片进行精修到对证件照的“换底”。在电商行业中，通过抠图技术将商品更加完美的展现给用户。这些需求都在促进着抠图技术向着高效率和高精度更近一步，但也给其带来了巨大的挑战和更高水平的要求。

由于应用需求的提高，抠图技术也受到广大研究人员的青睐。作为高精度的分割，抠图不仅需要前景目标的精确提取，更需要对前景混合像素部分有更好的表达。目前通过使用辅助图来简化抠图任务已经成为抠图算法的普遍共识，无论传统抠图算法还是基于深度学习的抠图算法都是使用辅助图作为先验信息输入到网络，如下图1所示。虽然辅助图的引入在一定程度上简化了抠图任务，使算法可以专注于

求解边界的半透明度。但是人工标注的辅助图也成为抠图技术泛化性能较差的原因。

(a)原图　　(b)三分图 trimap　　(c) Alpha 结果

图 1　使用三分图的抠图方法

为了使得抠图的应用场景可以进一步扩展，不使用辅助图的端化抠图算法受到研究人员的广泛研究。使用单张图片作为输入即可精确地提取前景的方法必然会推动图像抠图技术的再进步，使其应用到更加广泛的场景上，给生产生活带来诸多的便利。同时，在视频抠图领域也会得到更大发展。

2 相关研究技术

针对Image Matting这样一个基本的计算机视觉问题，复杂图片的抠图技术主要分为经典的传统方法和近年来兴起的深度学习方法。在传统方法部分，有三种经典算法分别为：通过采样方法的Bayes Matting[3]、传播方法的Closed Formed Matting[4]、根据非局部原理的KNN Matting[5]。而在深度学习方面，突出的有采用CNN的Deep Image Matting方法[6]。

2.1 Image Matting 基础介绍

抠图是一项从图片中将目标前景高精度提取出来的图像处理技术，一个典型的抠图过程如下图2所示，首先将图像分解为前景、

背景和未确定位置区域三个部分；接着通过传统方法或深度学习的方法生成Alpha图像，即精确区分前景与后景的Alpha矩阵；最后将其应用到其他场景背景中。

图 2　常见抠图过程

图像和视频中的前景准确估计问题，在实际应用中具有十分重要的意义。它是图像编辑和电影制作中的一项关键技术。1996年Alvy Ray Smith等人的《*Blue Screen Matting*》[1]，正式定义了Image Matting问题，即核心公式为

$$I = \alpha F + (1-\alpha)B$$

即给定一张图片I，可以将它分解成通过透明度α线性合成的前景F和背景B形式。抠图问题研究的是，如何通过左边的I，推测出右边的三个变量α、F和B。对于一张彩色图片来说，像素i位置上的RGB是已知的变量，背景B和前景F和α是未知的，因此上式中只有3个已知变量，却有着3+3+1=7个未知变量，难度可想而知。因此，传统的m抠图手段经常需要借助手工设计的三色图trimap作为额外的约束。

2.2 传统经典方法

1996年Alvy Ray Smith等人[1]给出了Triangulation Matting的方法。此方法进行了思路转换，将问题条件放松，假设知道了B和I，

那么有没有可能得到α和F，于是作者提出针对同一张前景切换背景，应用最小二乘法计算得到对应的透明度矩阵α和后景。

2004年Jian Sun等人的《*Poisson matting*》[2]是第一次在Image Matting课题中提出使用Trimap作为辅助工具。Trimap分为三种颜色，黑色代表完全背景，此处α为0；白色代表完全前景，此处α为1；灰色代表不确定区域，此处α为0.5。由于前面提到过方程是无确定解的，因此人工标注的Trimap图片就相当于约束条件了，并简化计算。

2003年Chuang Y Y等人提出贝叶斯抠图（Bayes Matting）[3]算法，核心是通过领域采样，从待定像素的领域像素中采样关联像素，利用图像像素点的颜色值特征，以定向高斯模型建立像素颜色模型，再用极大似然发估计α值。贝叶斯方法从条件概率的角度去考虑抠图问题，以贝叶斯公式建立F、B和α的联合概率分布，抠图问题就可以被转化为已知图片像素颜色I的情况下，为了最大化后验概率$P(F,B,\alpha|I)$，求F、B和α估计值的问题。

2006年Levin A等提出封闭式抠图（Closed Formed Matting）[4]，核心思想是通过传播的方法来利用图像的颜色特征，即允许α值从已知区域传播到邻域未知域。封闭式抠图基于颜色分布遵循颜色线性模型和前景背景颜色值局部平滑两个假设，并由此推知未知像素的α值和该像素点的颜色I呈线性相关关系，经过矩阵计算之后，代价函数表达式转化为关于α的二次型形式：

$$J(\alpha)=\alpha^T L\alpha$$

其中L为拉普拉斯矩阵（matting Laplacian matirx），可以由图像局部像素点颜色计算得到。由此，待定点的α值可以在没有明确估计该像素点前景和背景颜色的情况下以封闭的形式求解。

封闭式抠图方法提出的拉普拉斯矩阵被广泛地于各种抠图方法中。例如，K近邻抠图（KNN Matting）[5]就是通过在特征空间取k个最邻近点计算得拉普拉斯矩阵计算α值，这种非局部抠图关注于可以处理更稀疏的trimap，甚至不需要附加的约束条件输入，直接通过图像自身的前景与背景属性的区别来进行分割。

2.3 深度学习方法

传统Alpha Matting方法通常只使用低层次的特征，而深度学习算法可以考虑到更高层次的图像特征，并且可以通过创建大型图像数据集来自动优化算法。随着深度学习在近十年的研究突破，2017年Xu N等人提出一种基于深度学习的新抠图算法（Deep Image Matting）[6]，使用的CNN模型分为两个阶段，如下图3所示：首先利用深度卷积编码器将原图和对应的trimap图作为输入，预测图像的α；接着是一个小型卷积神经网络，对第一个网络的输出进行精修优化。由于CNN网络可以捕捉到更高阶的图像结构和语义特征，并且可以通过大数据集进行优化，基于深度学习的抠图方法可以更好的适用于真实图像。

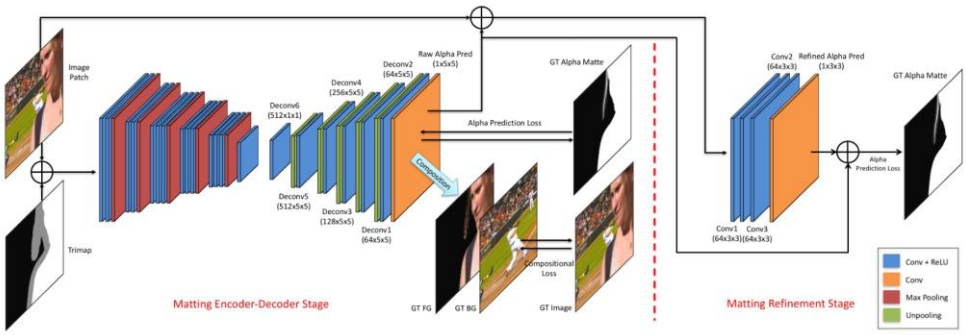

图 3　Deep Image Matting 由 enconder-decoder stage 和 refinement stage 构成

3 具体实现过程

3.1 前景图和合成图

我主要根据三种传统经典方法，实现了Closed Formed Matting[4]、KNN Matting[5]和Bayes Matting[3]三种算法。对于每种方法，均有统一的裁剪流程，首先我利用原图与trimap图片计算获取出alpha矩阵，再利用alpha矩阵裁剪出前景矩阵，再结合alpha图片和前景图获取前景图片，具体代码如下，

```
# 裁剪出前景图
def cutout(title):
    scale = 1.0
    image = load_image(image_path, "RGB", scale)  # 加载原图
    trimap = load_image(trimap_path, "GRAY", scale)  # 加载
     trimap 图片
    # 利用 image 和 trimap 估计 alpha 透明度矩阵
    if title=="cf":
        alpha = estimate_alpha_cf(image, trimap)
    elif title=="knn":
        alpha = estimate_alpha_knn(image, trimap)
    elif title=="lbdm":
        alpha = estimate_alpha_lbdm(image, trimap)
    # 根据 alpha 获取前景图
    foreground = estimate_foreground_ml(image, alpha)
    # 根据前景图和 alpha 图进行裁剪
```

```
16.     cutout = stack_images(foreground, alpha)
17.     # 保存图片
18.     save_image(cutout_path+title+"_cutout.png", cutout)
```

这一裁减后的输出图片如下所示。

图 4　Closed Formed Matting 算法生成前景图

图 5　KNN Matting 算法生成前景图

图 6　Bayes Matting 算法生成前景图

为了体现出抠图的主要目标——替换背景，我尝试加入四川大学校门处的背景图片，并对生成图片进行融合，代码如下，

```
1. def grid():
2.     scale = 1.0
```

```
    # 加载原图
    image = load_image(image_path, "RGB", scale, "box")
    # 加载 trimap 图片
    trimap = load_image(trimap_path, "GRAY", scale, "nearest")
    # 加载背景图
    new_background = load_image(background_path, "RGB", scale, "box")
    # 利用 image 和 trimap 估计 alpha 透明度矩阵
        alpha_cf = estimate_alpha_cf(image, trimap)
        alpha_knn = estimate_alpha_knn(image, trimap)
        alpha_lbdm = estimate_alpha_lbdm(image, trimap)
        # 根据 alpha 获取前景图和后景图
        foreground_cf = estimate_foreground_ml(image, alpha_cf)
        foreground_knn = estimate_foreground_ml(image, alpha_knn)
        foreground_lbdm = estimate_foreground_ml(image, alpha_lbdm)
        # 利用 alpha 矩阵对前后背景按照公式进行合并
        new_image_cf = blend(foreground_cf, new_background, alpha_cf)
        new_image_knn = blend(foreground_knn, new_background, alpha_knn)
        new_image_lbdm = blend(foreground_lbdm, new_background, alpha_lbdm)
        # 将换背景图保存在四方格中
        images = [image, trimap, alpha_cf, new_image_cf, alpha_knn, new_image_knn, alpha_lbdm, new_image_lbdm]
        grid = make_grid(images, 2, 4)
        save_image(grid_path, grid)
```

这一融合后的图片如下所示，其中第一列从第二行其分别是Closed Formed Matting、KNN Matting和Bayes Matting计算得到的alpha矩阵。

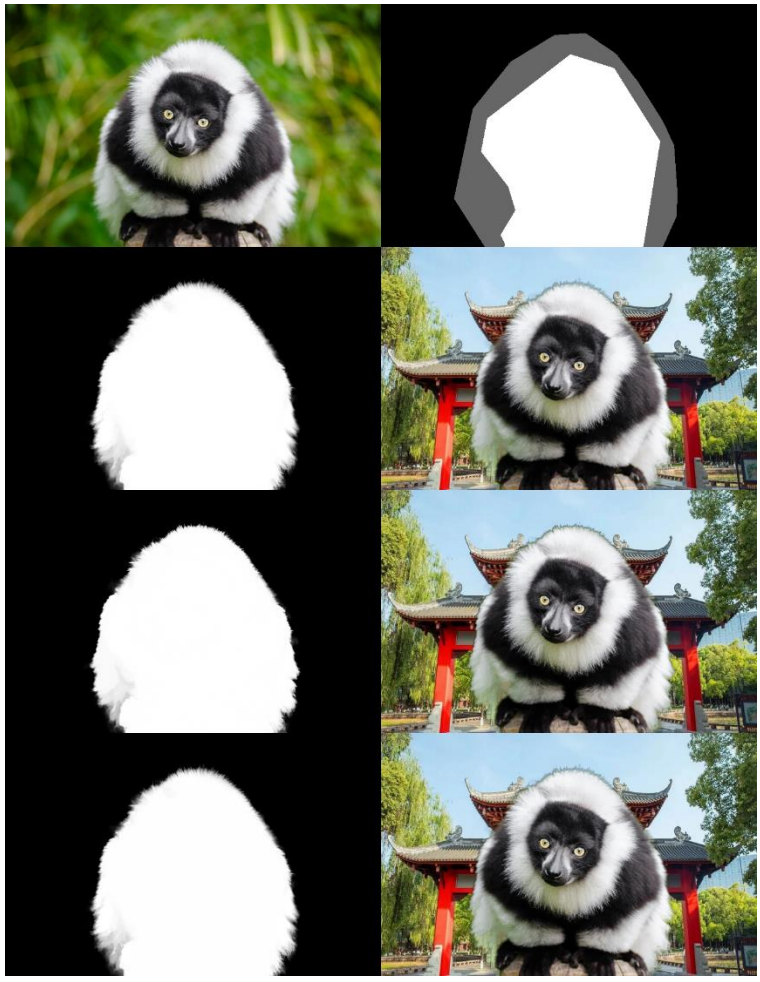

图 7　三类算法比较替换背景图

3.2 三类算法具体代码

针对三种算法，我将分别展示主要代码。

对于Closed Formed Matting方法，首先识别trimap图像确定前景和背景，根据Closed Formed方法计算对应拉普拉斯系数矩阵，最后再用共轭梯度法求解目标方程，得到最终alpha矩阵的解，具体代码如下，

```
1.  def estimate_alpha_cf(image, trimap, preconditioner=ichol, laplac
        ian_kwargs={}, cg_kwargs={}):
2.      # 确定预调节器为 ichol
3.      # 加载识别 trimap 图像
```

```
    is_foreground, is_background, is_known, is_unknown = trimap_split(trimap)
    # 计算拉普拉斯矩阵作为系数
    L = cf_laplacian(image, **laplacian_kwargs, is_known=is_known)
    L_U = L[is_unknown, :][:, is_unknown]
    R = L[is_unknown, :][:, is_known]
    m = is_foreground[is_known]
    # 对 trimap 副本降维至一维
    x = trimap.copy().flatten()
    # 共轭梯度法求解线性方程组 L_U * x = -R.dot(m)
    x[is_unknown] = cg(L_U, -R.dot(m), M=preconditioner(L_U), **cg_kwargs)
    # 约束 alpha 矩阵数值在[0,1]之间
    alpha = np.clip(x, 0, 1).reshape(trimap.shape)
    return alpha
```

对于KNN Matting方法，首先通过拉普拉斯矩阵计算得到线性方程组 $A=L+\lambda\cdot C$ 的系数 A，再利用共轭梯度法求解线性方程组 $A\cdot C=b$ 得到alpha矩阵的解，具体代码如下

```
def estimate_alpha_knn(image, trimap, preconditioner=jacobi, laplacian_kwargs={}, cg_kwargs={}):
    # 确定预调节器为 jacobi
    # 计算线性方程组系数 A, b
    A, b = make_linear_system(knn_laplacian(image, **laplacian_kwargs), trimap)
    # 共轭梯度法求解线性方程组 A * x = b
    x = cg(A, b, M=preconditioner(A), **cg_kwargs)
    # 约束 alpha 矩阵数值在[0,1]之间
    alpha = np.clip(x, 0, 1).reshape(trimap.shape)
    return alpha
```

对于Bayes Matting方法，与KNN方法类似，具体代码如下

```
def estimate_alpha_lbdm(image, trimap, preconditioner=ichol, laplacian_kwargs={}, cg_kwargs={}):
    # 确定预调节器为 ichol
    # # 计算线性方程组系数 A, b
    A, b = make_linear_system(lbdm_laplacian(image, **laplacian_kwargs), trimap)
```

```
    # 共轭梯度法求解线性方程组 A*x = b
    x = cg(A, b, M=preconditioner(A), **cg_kwargs)
    # 约束 alpha 矩阵数值在[0,1]之间
    alpha = np.clip(x, 0, 1).reshape(trimap.shape)
    return alpha
```

3.3 Laplacian 矩阵算法举例

根据论文[4]，可以推出需要优化的目标函数为

$$J(\alpha)=\sum_{i=1}^{n}\bar{\alpha}_k^{\ T}L_k\bar{\alpha}_k$$

而根据构造，可以得到$J(\alpha)=\alpha^T L\alpha$，其中$L$又被称为拉普拉斯矩阵，从而优化目标为

$$\min_\alpha\ \alpha^T L\alpha$$
$$s.t.\ \alpha_i=0\ for\ i\in BG,\ \alpha_i=1\ for\ i\in FG$$

其中BG是背景索引集合，FG是前景索引集合。以Closed Formed Matting为例，以下代码展示了求解alpha矩阵，构建拉普拉斯矩阵的详细过程。

```
def _cf_laplacian(image, epsilon, r, values, indices, indptr, is_
    known):
    h, w, d = image.shape
    assert d == 3
    size = 2 * r + 1
    window_area = size * size
    for yi in range(h):
        for xi in range(w):
            i = xi + yi * w
            k = i * (4 * r + 1) ** 2
            for yj in range(yi - 2 * r, yi + 2 * r + 1):
                for xj in range(xi - 2 * r, xi + 2 * r + 1):
                    j = xj + yj * w
                    if 0 <= xj < w and 0 <= yj < h:
                        indices[k] = j
                    k += 1
```

```
            indptr[i + 1] = k
    # 居中并归一化窗口颜色
    c = np.zeros((2 * r + 1, 2 * r + 1, 3))
    # 对图片的每个像素
    for y in range(r, h - r):
        for x in range(r, w - r):
            if np.all(is_known[y - r : y + r + 1, x - r : x + r + 1]):
                continue
            # 对每个颜色通道
            for dc in range(3):
                # 计算窗口中颜色通道的总和
                s = 0.0
                for dy in range(size):
                    for dx in range(size):
                        s += image[y + dy - r, x + dx - r, dc]
                # 计算居中的窗口颜色
                for dy in range(2 * r + 1):
                    for dx in range(2 * r + 1):
                        c[dy, dx, dc] = (
                            image[y + dy - r, x + dx - r, dc] - s / window_area
                        )
            # 通过正则化计算颜色通道上的协方差矩阵
            a00, a01, a02 = epsilon, 0.0, 0.0
            a11, a12, a22 = epsilon, 0.0, epsilon
            for dy in range(size):
                for dx in range(size):
                    a00 += c[dy, dx, 0] * c[dy, dx, 0]
                    a01 += c[dy, dx, 0] * c[dy, dx, 1]
                    a02 += c[dy, dx, 0] * c[dy, dx, 2]
                    a11 += c[dy, dx, 1] * c[dy, dx, 1]
                    a12 += c[dy, dx, 1] * c[dy, dx, 2]
                    a22 += c[dy, dx, 2] * c[dy, dx, 2]
            a00 /= window_area
            a01 /= window_area
            a02 /= window_area
            a11 /= window_area
            a12 /= window_area
            a22 /= window_area
            # 行列式
```

```
55.             det = (a00*a12*a12 + a01*a01*a22 + a02*a02*a11 - a00*
        a11*a22 - 2 * a01*a02*a12)
56.             inv_det = 1.0 / det
57.             # 计算协方差矩阵
58.             m00 = (a12 * a12 - a11 * a22) * inv_det
59.             m01 = (a01 * a22 - a02 * a12) * inv_det
60.             m02 = (a02 * a11 - a01 * a12) * inv_det
61.             m11 = (a02 * a02 - a00 * a22) * inv_det
62.             m12 = (a00 * a12 - a01 * a02) * inv_det
63.             m22 = (a01 * a01 - a00 * a11) * inv_det
64.             # 对在(2r + 1)*(2r + 1)窗口内的每个像素对
        ((xi, yi), (xj, yj))
65.             for dyi in range(2 * r + 1):
66.                 for dxi in range(2 * r + 1):
67.                     s = c[dyi, dxi, 0]
68.                     t = c[dyi, dxi, 1]
69.                     u = c[dyi, dxi, 2]
70.                     c0 = m00 * s + m01 * t + m02 * u
71.                     c1 = m01 * s + m11 * t + m12 * u
72.                     c2 = m02 * s + m12 * t + m22 * u
73.                     for dyj in range(2 * r + 1):
74.                         for dxj in range(2 * r + 1):
75.                             xi = x + dxi - r
76.                             yi = y + dyi - r
77.                             xj = x + dxj - r
78.                             yj = y + dyj - r
79.                             i = xi + yi * w
80.                             j = xj + yj * w
81.                             # 计算对像素对每个 L_ij 贡献
82.                             temp = (c0*c[dyj, dxj, 0] + c1*c[dyj,
         dxj, 1] + c2*c[dyj, dxj, 2])
83.                             value = (1.0 if (i == j) else 0.0) -
        (1 + temp) / window_area
84.                             dx = xj - xi + 2 * r
85.                             dy = yj - yi + 2 * r
86.                             values[i, dy, dx] += value
87.
88. def cf_laplacian(image, epsilon=1e-7, radius=1, is_known=None):
89.     h, w, d = image.shape
90.     n = h * w
91.     if is_known is None:
```

```
92.          is_known = np.zeros((h, w), dtype=np.bool8)
93.      is_known = is_known.reshape(h, w)
94.      # Data for matting laplacian in csr format
95.      # 游标指针
96.      indptr = np.zeros(n + 1, dtype=np.int64)
97.      # 列索引
98.      indices = np.zeros(n * (4 * radius + 1) ** 2, dtype=np.int64)
99.      values = np.zeros((n, 4 * radius + 1, 4 * radius + 1), dtype=
           np.float64)
100.     # 详细计算
101.     _cf_laplacian(image, epsilon, radius, values, indices, indptr
           , is_known)
102.     # 列稀疏矩阵
103.     L = scipy.sparse.csr_matrix((values.ravel(), indices, indptr)
           , (n, n))
104.     return L
```

对于KNN和Bayes算法的拉普拉斯矩阵与此类似。

4 环境配置及说明

此次我在AutoDL的线上环境平台的实验环境中，配置了PyTorch1.11.10、Python3.8、Cuda 11.3的镜像环境。

图 8　镜像环境总结

在CPU环境下，需要安装的环境如下

```
1.  numpy>=1.16.0
2.  pillow>=5.2.0
3.  numba>=0.47.0
4.  scipy>=1.1.0
```

在GPU环境下，还需要安装的环境如下

```
1.  cupy-cuda90>=6.5.0 or similar
2.  pyopencl>=2019.1.2
```

在安装此环境后，可直接运行如下命令，即可得到前景图片和换背景图片

```
1.  python -u cutout.py
2.  python -u grid.py
```

5 结果分析

分别查看三种算法得到的alpha矩阵(如图7），我们可以发现在细节方面三种方法还是有一定区别的。重点关注左下角的毛绒，可以发现KNN算法对于细节的体现更加清晰。通过定性分析，发现三种方法均可以精确地提取图像中的前景目标，在边界部分去的较好的处理效果。

6 本课程设计收获或对未来的展望

在充斥着图像信息的数字时代，图像处理技术已经成为不可或缺的一部分，而抠图技术更是其中的关键一环，在诸多应用场景中都有着广泛的应用。我在本课程设计中，针对三种传统算法进行了尝试，从对算法拉普拉斯矩阵的计算，到对alpha图像的提取，最终在顺利提取出前景图和替换背景。这一系列尝试不仅使我的代码能力得到提高，还使我对数字图像处理有了更深的理解和深入的尝试。

随着抠图技术的更新升级，各类方法的迭代使得传统方法各方面性能都相形见绌，但是现有算法显露的弊端也在阻碍着抠图技术的使用，如在现有传统方法中运行时间过高，在深度学习中无法摆脱辅助图的使用等。在我本设计中所深入的alpha抠图方面，辅助图的应用是一大进步。

应用辅助图的目的是为了确定目标在图像中的位置和简化抠图任务，我认为此目的可通过将抠图目标分解的方式实现，从而避免使用辅助图。随着技术的不断改进，抠图技术可以加入识别算法，首先确定图片中各物体位置，再通过简单的图形标注确定抠图目标，并自动进行精确细化的抠图计算。

7 参考文献

[1] Alvy Ray Smith, James F. Blinn. Blue screen matting. international conference on computer graphics and interactive techniques, 1996.

[2] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson Matting. International conference on computer graphics and interactive techniques, Aug 2004.

[3] Yung-Yu Chuang, Brian Curless, David Salesin, and Richard Szeliski. A Bayesian approach to digital matting. 2003.

[4] Anat Levin, Dani Lischinski, and Yair Weiss. A closed-form solution to natural image matting. IEEE transactions on pattern analysis and machine intelligence, 30(2):228–242, 2007.

[5] Qifeng Chen, Dingzeyu Li, and Chi-Keung Tang. Knn matting. IEEE transactions on pattern analysis and machine intelligence, 35(9):2175–2188, 2013.

[6] Ning Xu, Brian Price, Scott Cohen, and Thomas S. Huang. Deep Image Matting. arXiv: Computer Vision and Pattern Recognition, 2017.