IMVFX Homework 1: KNN Matting Report

Zhi-Yi Chin
joycenerd.cs09@nycu.edu.tw

November 3, 2021

1 Introduction

Our goal is to apply KNN matting[1] to do image composition. We use KNN matting to get the α value from the original image with a corresponding trimap; we extract the foreground from the original image and paste it to a new background. The α value decides the pixel weight of the composite image.

2 Method

2.1 Find KNN for the given image

First, we get the index for all the positions in the image. The order of the position is from left to right and top to bottom. Then we split the positions into rows and columns. Rows save the y value of all the positions, and columns save the x value of all the positions.

Then we get the feature vector of the image.

$$X(i) = (R, G, B, x, y)_i \tag{1}$$

The feature vector splits into two parts, and we called it feat_1 and feat_2. Feat_1 consists of the pixel value of the image, which is the R, G, B in Eq. 1. We reshape the original image to $(h \times w, c)$, h is the height of the original image, w is the original image's width, and c is the channel of the image; for a color image, it is 3. Feat_2 is the x, y in Eq. 1. We implement this by using the rows and columns in the last step; also, we normalize the x, y by dividing by $\sqrt{h^2 + w^2}$. Then we combine feat_1 and feat_2 as the complete feature vector.

Next, we construct the nearest-neighbor model using scikit-learn and set the k to 10. We give the model the feature vector, and we will get k (10) nearest neighbors for each pixel.

```
# get the index of each position
all_pixs=np.arange(h*w)
rows,cols=np.unravel_index(all_pixs,(h,w))

# get feature vector
feat_1=image.reshape(h*w,c).T
idx_list=np.vstack((rows,cols))
feat_2=idx_list/np.sqrt(h*h+w*w)
```

```
feat_vec=np.append(feat_1,feat_2,axis=0).T

# find k nearest neighbors
model=sklearn.neighbors.NearestNeighbors(n_neighbors=10,n_jobs=3)
model.fit(feat_vec)
knn=model.kneighbors(feat_vec)[1]
```

Listing 1: Implementation of KNN.

2.2 Compute the affinity matrix A and all other stuff

First, we compute the kernel function $\mathcal{K}(i, j)$ for soft segmentation.

$$\mathcal{K}(i,j) = 1 - \frac{\|X(i) - X(j)\|}{C+2} \tag{2}$$

Eq. 2 is the kernel function we use. i is the row index, which is range from 0 to $h \times w$. j is the column index, which is KNN results from section 2.1. We then get the affinity matrix A using `scipy.sparse`, which we will get a sparse matrix in COOrdinate format. We need to use a sparse matrix because using the normal way to save the matrix will be out of memory. In our case, we have a large matrix with mostly 0, so using a sparse matrix is a memory-efficient way.

Next, we get the Laplacian L by $D - A$, which D is a diagonal matrix with a column sum of A as the diagonal values. Then we get another diagonal matrix M where $M = \text{diag}(m)$ and m is a binary vector of indices of all the marked-up pixels. To get m, we add the foreground mark and background mark together. Then we compute v, a binary vector of pixel indices corresponding to user markups for a given layer. In our case, the foreground is what we are interested in, so v is the foreground mark.

We have all the things we need. Now we can look at our objective function. Eq. 3 is our objective function. In the actual implementation, we multiply the objective function by a factor of 2.

$$\begin{aligned}(L + \lambda M)\alpha &= \lambda v \\ (L + \lambda M)\alpha - \lambda v\end{aligned} \tag{3}$$

```
# get affinity matrix A
row_idx=np.repeat(all_pixs,10)
col_idx=knn.reshape(h*w*10)
kernel=1-np.linalg.norm(feat_vec[row_idx]-feat_vec[col_idx],axis=1)/(c+2)
A=scipy.sparse.coo_matrix((kernel,(row_idx,col_idx)),shape=(h*w,h*w))

# get diagonal matrix D and L
diag_val=np.ravel(A.sum(axis=1))
D=scipy.sparse.diags(diag_val)
L=D-A

# get diagonal matrix M and assigned alpha v
tri_mark=np.ravel(foreground+background)
M=scipy.sparse.diags(tri_mark)
```

```
v=np.ravel(foreground)

# (L+M)-v
first=2*(L+my_lambda*M)
second=2*my_lambda*v.T
```

Listing 2: Implementation of affinity matrix A and the objective function.

2.3 Solve for the linear system

The objective function Eq. 3 is of the form $Ax-b$ so that we can use the same method, but to mind that we are dealing with a sparse matrix. We use `scipy.sparse.linalg.spsolve` to solve this sparse linear system if no warning occurs, otherwise, we use `scipy.sparse.linalg.lsqr` which is a least-square solution. In order to get the final solution, we need to ensure that our solution is larger or equal to 0 and smaller or equal to 1, so we clip the values to $[0, 1]$. The results are the α values for the foreground.

```
warnings.filterwarnings('error')
alpha = []
try:
    sol=scipy.sparse.linalg.spsolve(first,second)
    alpha=np.clip(sol,0,1).reshape(h,w)
except Warning:
    sol=scipy.sparse.linalg.lsqr(first,second)[0]
    alpha=np.clip(sol,0,1).reshape(h,w)
```

Listing 3: Implementation for solving the objective function.

2.4 Composition

Eq. 4 is how we composite the foreground image and the new background image. We then choose the new background image we want and resize it to the same size as the original background. C is the resulting image, F is the foreground image, and B is the new background image.

$$C = \alpha F + (1 - \alpha)B \tag{4}$$

```
[h,w,c]=image.shape
background=cv2.resize(background,(w,h),interpolation=cv2.INTER_AREA)

result = []
alpha_3d=np.repeat(alpha,repeats=3,axis=2)
result=alpha*image+(1-alpha)*background
cv2.imwrite(f'./result/{args.name}.png', result)
```

Listing 4: Implementation of image composition.

We have referenced [2] for our implementation.

3 Experimental Results

	bear	gandalf	woman	winnie (bonus)
original				
k=10				
k=50				
k=100				

Table 1: Qualitative matting results comparing the original image, $k = 10$, $k = 50$, and $k = 100$.

For the image bear, gandalf and winnie, $k = 10$ is already enough. For bear and winnie, no matter how k changes, the results are the same. We think this is because, in the original images of these two objects, the foreground and background are easy to separate (foreground object is a large complete area, background color, and foreground color has high contrast). For gandalf, if k increases, artifacts increase as well, so $k = 10$ is the best for gandalf; the result is shown in Figure 1. For woman, since the hair is difficult to segment, we find that the larger the k, the better the segmentation of the hair is, the result is shown in Figure 2. However, there are some artifacts on

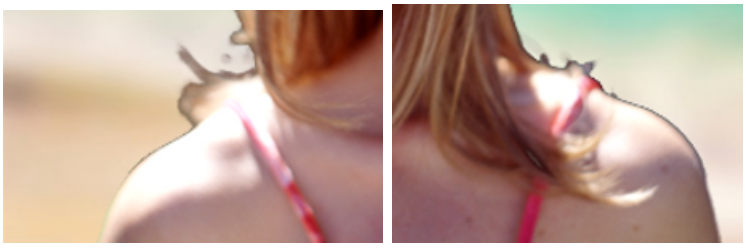
the hair ends on the shoulder. No matter how large the k is, the artifacts still exist; the result is shown in Figure 3.

(a) $k = 10$ zoom in. (b) $k = 50$ zoom in. (c) $k = 100$ zoom in.

Figure 1: The artifacts increase while k increase in the gandalf image.

(a) $k = 10$ zoom in. (b) $k = 50$ zoom in. (c) $k = 100$ zoom in.

Figure 2: The artifacts decrease while k increases in the woman image.



(a) (b)

Figure 3: For the woman image, hair ends on the shoulder does not segment well not matter how we tune our k.

4 Bonus

4.1 Apply to other images

For the bonus, we have tried our method on an image other than the images provided. We download an image from http://www.alphamatting.com/datasets.php. This dataset is provided by Alpha Matting [3] team. We choose the image that has winnie as the foreground. The results are shown in Table. 1.

4.2 Execution time comparison

If the image is smaller then we use `numpy` solver instead of `scipy.sparse`. Take gandalf, for example. When using `scipy.sparse` solver, the execution time is 71.54 seconds. When using `numpy` solver, then the execution time is 6.24 seconds. From the results, we can see that `numpy` solver is faster than `scipy.sparse` solver. The downside of `numpy` solver is, it cannot accept large images because it needs to process the whole matrix instead of a sparse form matrix. When processing large matrices, we need lots of memory to save the whole matrix, and we will soon get a memory error.

4.3 Gaussian kernel

We have also used other kernels to represent the feature. We applied the Gaussian kernel. Most of the composite image has a sharp foreground, sometimes too sharp in our opinion, so we applied the Gaussian kernel and hoped to get a smoother foreground. We test on woman and gandalf and set $k = 50$. Below is our qualitative result.

(a) gandalf (b) woman

Figure 4: Results of using the Gaussain kernel and set $k = 50$.

```
elif k_method=='gaussian':
    kernel=np.exp(-(feat_vec[row_idx]-feat_vec[col_idx])**2/sigma**2)
    kernel=np.sum(kernel,axis=1)/(c+2)
```

Listing 5: Implementation of the Guassian kernel.

References

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

[2] M. Forte and R. Childs. knn-matting, 2018. URL https://github.com/MarcoForte/knn-matting.

[3] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott. A perceptually motivated online benchmark for image matting. In *Proceddings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. URL http://publik.tuwien.ac.at/files/PubDat_180666.pdf. Posterpräsentation: IEEE Conference on Computer Vision and Pattern Recognition, 2009. CVPR '09, Miami, Florida, USA; 2009-06-20 – 2009-06-25.