Homework 1

1. Explanation of my code

這次作業實作的部分可以分成幾個部分

a. 找出每個像素最像**k**個像素:

```
def find_Knn_image(fi,K):
    print("Start to find the nearest Knn")
    h,w,c = fi.shape
    i, j = np.unravel_index(np.arange(h*w), (h, w))
    feature_vec = np.append(np.transpose(fi.reshape(h*w,c)), [ i, j]/np.sqrt(h*h + w*w), axis=0).T
    knn = NearestNeighbors(n_neighbors=K,n_jobs=8)
    knn.fit(feature_vec)
    distances, indices = knn.kneighbors(feature_vec)
    return indices,feature_vec
```

首先利用以下的式子，計算出每個像素相對於每個像素的差異:

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

$$A = \begin{bmatrix} k(1,1) & \cdots & k(1,n) \\ \vdots & \ddots & \vdots \\ k(n,1) & \cdots & k(n,n) \end{bmatrix}$$

再利用NearestNeighbors函式找出與各個像素最像的K個像素。

b. 再計算需要做**KNN Matting**的參數:

```
def computing_parameter(img,knn,K,feature_vec,constraints,foreground,mylambda):
    h,w,c = img.shape
    print("Start to compute the affinity matrix A and all other stuff needed")
    row_inds = np.repeat(np.arange(h*w),10)
    col_inds = knn.reshape(h*w*10)
    vals = 1 - np.linalg.norm(feature_vec[row_inds]-feature_vec[col_inds],axis=1)/(c+2)
    A = scipy.sparse.coo_matrix((vals, (row_inds, col_inds)),shape=(h*w, h*w))

    D_value = scipy.sparse.diags((np.ravel(A.sum(axis=1))))
    L = D_value-A
    M = scipy.sparse.diags(np.ravel(constraints[:,:]))
    v = np.ravel(foreground[:,:])
    c = 2*mylambda*np.transpose(v)
    H = 2*(L + mylambda*M)

    return H,c
```

根據老師的講義，產生需要的矩陣:

Evaluation for KNN matting (cont.)

- We want α of mark-up pixels fitting for users assignment.

$$\boldsymbol{M\alpha \approx v}$$

$\boldsymbol{\alpha}$: the vector of all α
$\boldsymbol{M}$: diag($\boldsymbol{m}$), indicates whether a pixel is marked-up.
$\boldsymbol{v}$: *indicates the assigned alpha*

- Besides, $\boldsymbol{D\alpha \approx A\alpha}$

$\boldsymbol{D}$: diag(D_i), an *N*x*N* diagonal matrix
$\boldsymbol{A}$: *N*x*N* affinity matrix composed of $k(i,j)$

- Our objective function becomes

$$Obj = (\boldsymbol{D - A})\boldsymbol{\alpha} + \lambda(\boldsymbol{M\alpha - v}) = (\boldsymbol{L} + \lambda\boldsymbol{M})\boldsymbol{\alpha} - \lambda\boldsymbol{v}$$

$\boldsymbol{L}$= ($\boldsymbol{D}$-$\boldsymbol{A}$)

λ: a weight controlling user's confidence on the markups

c. Solve for the linear system

We view the objective function as an Ax=b linear equation and solve it with x = A-1b

```
print("Start to compute through linear system")
warnings.filterwarnings('error')
alpha = []
try:
    alpha = np.minimum(np.maximum(scipy.sparse.linalg.spsolve(H, c), 0), 1).reshape(h, w)
except Warning:
    x = scipy.sparse.linalg.lsqr(H, c)
    alpha = np.minimum(np.maximum(x[0], 0), 1).reshape(h, w)

return alpha
```

d. Composite with other images:

```
background = cv2.imread('./image/Background.jpg')
alpha = np.expand_dims(alpha,axis=2)
alpha = np.repeat(alpha,3,axis=2)
background[:alpha.shape[0],:alpha.shape[1]] = np.multiply(image[:alpha.shape[0],:alpha.shape[1]],alpha)+ \
                                              np.multiply((1-alpha),background[:alpha.shape[0],:alpha.shape[1]])
# result = []
cv2.imwrite(f'./result/{target}.png', background)
```

因為合成的照片與背景大小不同，所以我找了比前景照片還要大的照片當背景，這樣可以把前景照片跟左下角對齊就可以合成照片。

2. Experiment

在這個作業我做了兩個實驗:

1. 改變Lambda值
2. 改變KNN中K的數量

改變**Lamda值:**

在講義中，有提到Lamda是對markup的信心，我想觀察在不同值下對結果的影響。

Lamda=100:

Lamda=10000:

Lamda=1:

因為MarkUp的地方很理想，所以可以看出Lamda的值基本上對結果沒有甚麼影響。

改變**KNN**中**K**的數量**:**

K=1:

K=5:

K=10:

K=20:

可以看出K的數量在超過一定的值後影響結果不大，而在K值比較小的情況下效過就會明顯比較差

3. Bonus:

這次的作業中，我做了幾項Bonus:

1. 減少運算時間
2. 改進feature_vec

減少運算時間:

這次的作業在運算alpha的最後一步驟中，需要比較大的計算量，而剛好scipy.sparse有專門的函數可以幫助稀疏矩陣的運算。

改進**feature_vec:**

目前已有許多研究證實，相片在空間中具有連續性，所以我決定將距離當作attribute來計算各個pixel之間的差異。

(R,G,B) -> (R,G,B,w,h)

4. Citation:

dingzeyuli/knn-matting: Source Code for KNN Matting, CVPR 2012 / TPAMI 2013. MATLAB code ready to run. Simple and robust implementation under 40 lines. (github.com)

MarcoForte/knn-matting: Python implementation of KNN Matting, CVPR 2012 / TPAMI 2013 http://dingzeyu.li/projects/knn/ (github.com)