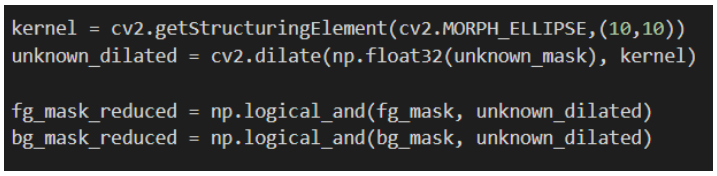
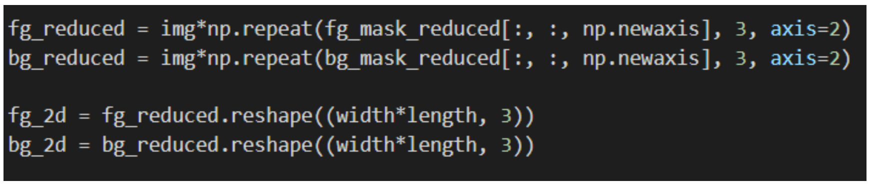
Part 1 : Explanation of my code

Dilate: Because we want the GMM to train faster, so we only want to use the points around the unknown part of the trimap. I use *cv2.dilate* to expand the unknown area and get unknown_dilate. If we logical and the unknown_dilate and fg_mask, we can get fg points around the unknown part. Same concept can be applied to bg too.



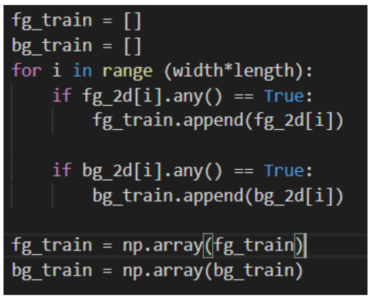
```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(10,10))
unknown_dilated = cv2.dilate(np.float32(unknown_mask), kernel)

fg_mask_reduced = np.logical_and(fg_mask, unknown_dilated)
bg_mask_reduced = np.logical_and(bg_mask, unknown_dilated)
```

Prepare ndarray to fit GMM-1: First step is to getting the color value of the reduced fg and bg points. Then turn it into the shape so that it can fit into GMM model.



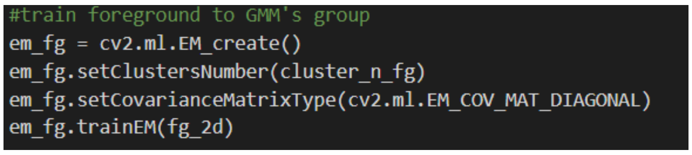
```
fg_reduced = img*np.repeat(fg_mask_reduced[:, :, np.newaxis], 3, axis=2)
bg_reduced = img*np.repeat(bg_mask_reduced[:, :, np.newaxis], 3, axis=2)

fg_2d = fg_reduced.reshape((width*length, 3))
bg_2d = bg_reduced.reshape((width*length, 3))
```

Prepare ndarray to fit GMM-2: We need to eliminate the <0,0,0> value points in fg-value and bg-value since they do not need to be put into GMM model.

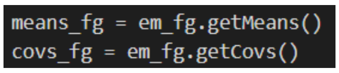


```
fg_train = []
bg_train = []
for i in range (width*length):
    if fg_2d[i].any() == True:
        fg_train.append(fg_2d[i])

    if bg_2d[i].any() == True:
        bg_train.append(bg_2d[i])

fg_train = np.array(fg_train)
bg_train = np.array(bg_train)
```

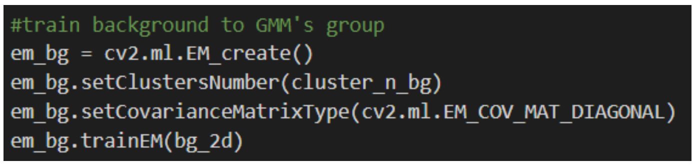
GMM training & getting means, covariance matrix for each clusters: Applied to both foreground and background.



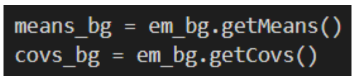
```
#train foreground to GMM's group
em_fg = cv2.ml.EM_create()
em_fg.setClustersNumber(cluster_n_fg)
em_fg.setCovarianceMatrixType(cv2.ml.EM_COV_MAT_DIAGONAL)
em_fg.trainEM(fg_2d)
```



```
means_fg = em_fg.getMeans()
covs_fg = em_fg.getCovs()
```



```
#train background to GMM's group
em_bg = cv2.ml.EM_create()
em_bg.setClustersNumber(cluster_n_bg)
em_bg.setCovarianceMatrixType(cv2.ml.EM_COV_MAT_DIAGONAL)
em_bg.trainEM(bg_2d)
```

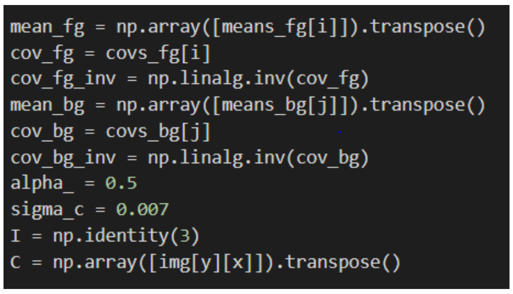


```
means_bg = em_bg.getMeans()
covs_bg = em_bg.getCovs()
```

Optimize alpha for each combination of clusters:

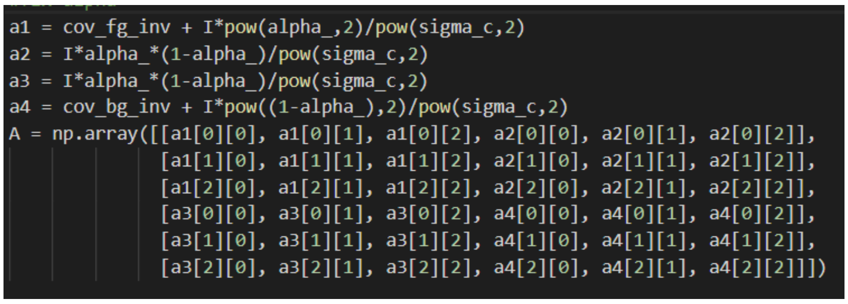
In each pair of foreground and background(the ith cluster of foreground and the jth cluster of background), I will do the following.

- Initialized all values

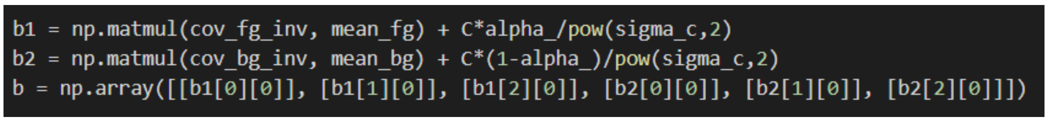


```
mean_fg = np.array([means_fg[i]]).transpose()
cov_fg = covs_fg[i]
cov_fg_inv = np.linalg.inv(cov_fg)
mean_bg = np.array([means_bg[j]]).transpose()
cov_bg = covs_bg[j]
cov_bg_inv = np.linalg.inv(cov_bg)
alpha_ = 0.5
sigma_c = 0.007
I = np.identity(3)
C = np.array([img[y][x]]).transpose()
```

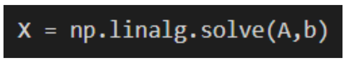
- To make sure the convergence of alpha : I run the optimization step for 200 loops
- In fix alpha step, we need to know the x in equation Ax=b, first we calculate the A and b. Then use *np.linalg.solve(A,b)* to get the value of x.



```
a1 = cov_fg_inv + I*pow(alpha_,2)/pow(sigma_c,2)
a2 = I*alpha_*(1-alpha_)/pow(sigma_c,2)
a3 = I*alpha_*(1-alpha_)/pow(sigma_c,2)
a4 = cov_bg_inv + I*pow((1-alpha_),2)/pow(sigma_c,2)
A = np.array([[a1[0][0], a1[0][1], a1[0][2], a2[0][0], a2[0][1], a2[0][2]],
              [a1[1][0], a1[1][1], a1[1][2], a2[1][0], a2[1][1], a2[1][2]],
              [a1[2][0], a1[2][1], a1[2][2], a2[2][0], a2[2][1], a2[2][2]],
              [a3[0][0], a3[0][1], a3[0][2], a4[0][0], a4[0][1], a4[0][2]],
              [a3[1][0], a3[1][1], a3[1][2], a4[1][0], a4[1][1], a4[1][2]],
              [a3[2][0], a3[2][1], a3[2][2], a4[2][0], a4[2][1], a4[2][2]]])
```

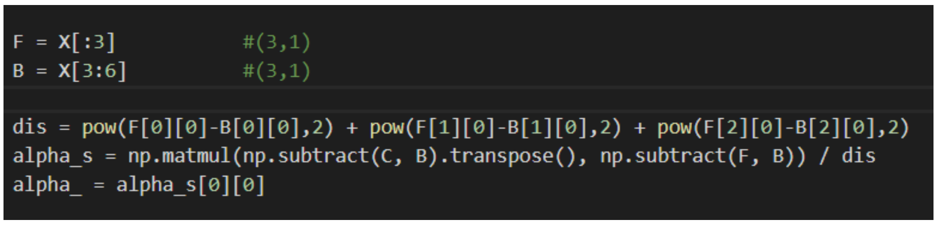


```
b1 = np.matmul(cov_fg_inv, mean_fg) + C*alpha_/pow(sigma_c,2)
b2 = np.matmul(cov_bg_inv, mean_bg) + C*(1-alpha_)/pow(sigma_c,2)
b = np.array([[b1[0][0]], [b1[1][0]], [b1[2][0]], [b2[0][0]], [b2[1][0]], [b2[2][0]]])
```



```
X = np.linalg.solve(A,b)
```

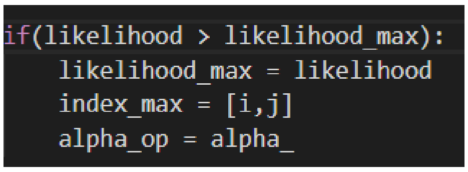
- In fix F and B step, we can use the F and B just calculated to get the value of alpha. (It's just the formula)



```
F = X[:3]           #(3,1)
B = X[3:6]          #(3,1)

dis = pow(F[0][0]-B[0][0],2) + pow(F[1][0]-B[1][0],2) + pow(F[2][0]-B[2][0],2)
alpha_s = np.matmul(np.subtract(C, B).transpose(), np.subtract(F, B)) / dis
alpha_ = alpha_s[0][0]
```

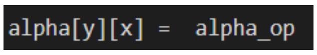
Chose the best likelihood of alpha : After each round of calculation, we can see if the alpha is better then the previous alpha. (From the likelihood formula)

The calculation of likelihood is too long for me to show the screenshot here.



```
if(likelihood > likelihood_max):
    likelihood_max = likelihood
    index_max = [i,j]
    alpha_op = alpha_
```

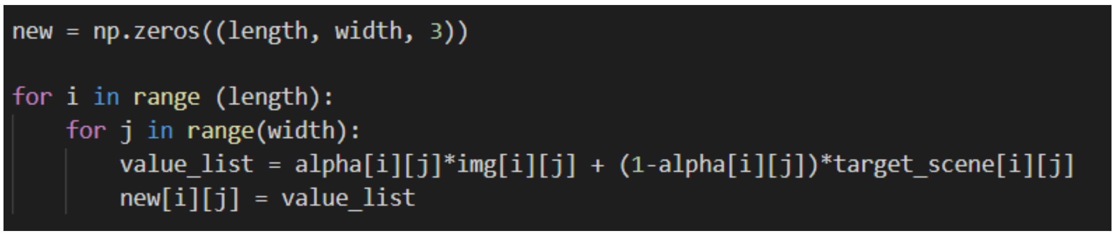
Alpha value is confirmed: After all pair of foreground and background, we can finally be sure of the alpha value and can fill it into the alpha array



```
alpha[y][x] =  alpha_op
```

Attach the foreground to new background :

C[i][j] = alpha*F[i][j] + (1-alpha)*B[i][j]



```
new = np.zeros((length, width, 3))

for i in range (length):
    for j in range(width):
        value_list = alpha[i][j]*img[i][j] + (1-alpha[i][j])*target_scene[i][j]
        new[i][j] = value_list
```

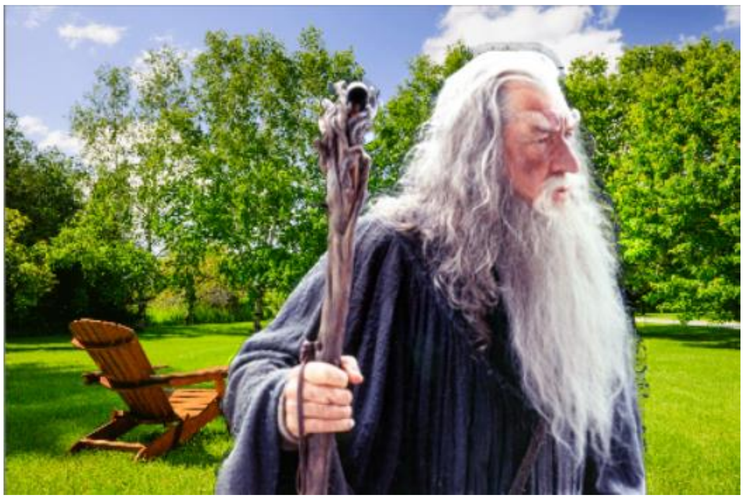
Part 2 : Experiment and Result

Gandalf:

- Parameter

Sigma_c	Dilate size	Background cluster#	Foreground cluster#	Loop # for converge
0.01	(50,50)	1	3	200

- Result

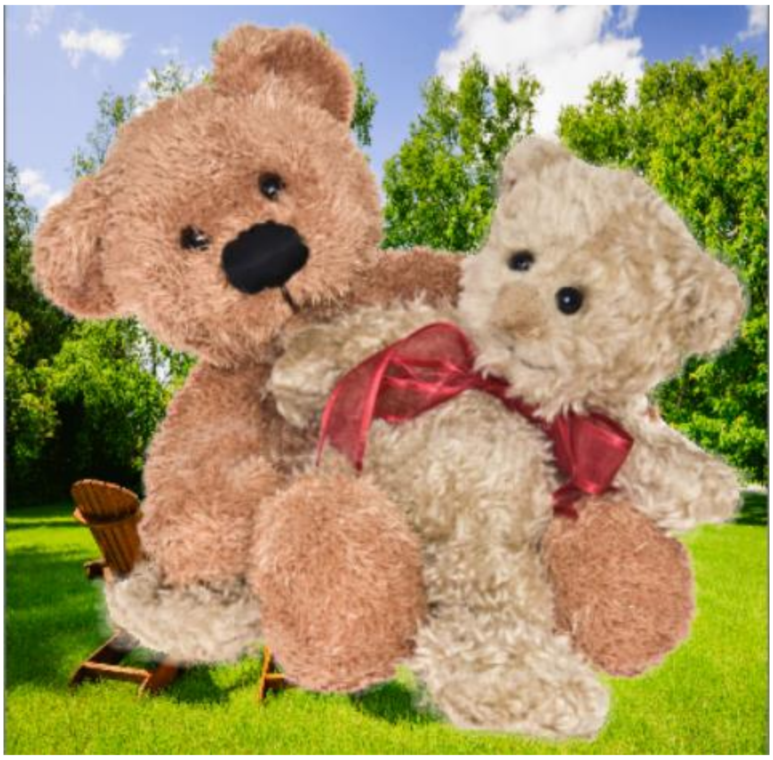


Bear:

- parameter

Sigma_c	Dilate size	Background cluster#	Foreground cluster#	Loop # for converge
0.01	(50,50)	1	2	200

- result

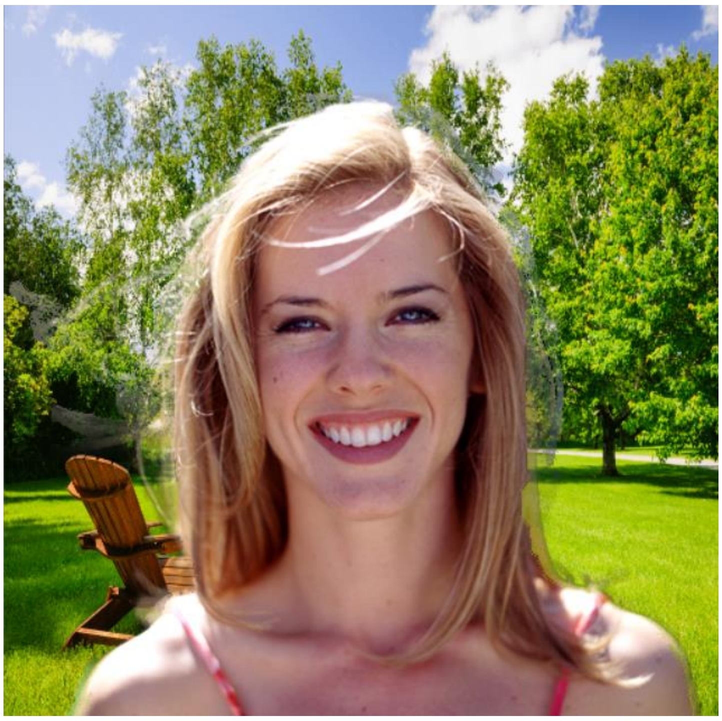


Woman:

- parameter

Sigma_c	Dilate size	Background cluster#	Foreground cluster#	Loop # for converge
0.01	(50,50)	3	2	200

- result



Conclusion:

First of all, very hard to debug. Don’t know which part of the calculation goes wrong. What we can do is to check every part of the code. Also, I have tried different parameters for many times. In my trials, I found out that my loop number for convergence is not big enough, so I have change it to a larger number. The result of “woman” and “gandalf” are not well, but the “bear” is fine.