APPM 4058A: Digital Image Processing

Project: Skin Detection

Tshepo Nkambule (1611821)

17 June 2020

1 Introduction

Image segmentation is the process of isolating certain objects of interests from a digital image. This means that image segmentation aims to assign labels to pixels in an image such that pixels with similar characteristics have the same label. The applications of image segmentation vary from wanting to identify unique objects in an image, find a particular object or extract certain details from an image for further exploration. One of the tasks in image segmentation is skin detection. In the task of skin detection the purpose is to segment object-s/parts of a colour image that are human skin from those that are not human skin. Skin detection has several applications such as in medicine to detect body limbs, gesture detection, pedestrian tracing and facial recognition.

2 Dataset

In order to perform skin detection a dataset is required. The main dataset used in this project is the **SFA** dataset adapted from www.sel.eesc.usp.br. This is a human skin database constructed based on face images. It consists of human skin samples (3 354) and non-skin samples (5 590) that were derived from the face images in the dataset. The samples are available in a number of dimensions and the ones used in this project are the 35×35 image samples. The face images in the dataset were also used for testing the algorithms that were implemented. Family photos adapted from cs-chan.com dataset were also used to test how the implemented algorithms perform on non face only images.

3 Skin detection

3.1 Colour spaces and Data loading

Skin colour is a discriminating factor that can be used to distinguish between skin and non-skin areas in an image. The default colour space for digital images

is the **RGB** colour space. In **RGB** colour space the luminance is embedded in all three channels. Luminance can vary across skin areas in an image and it is not reliable for separating skin regions from non-skin regions. This means that **RGB** colour space is not convenient for the skin detection task as luminance is captured across all channels while it is irrelevant. The first step is to find a suitable colourspace in which luminance can be easily discarded.

In **YCbCr** colour space the luminance is only concentrated on the **Y** component. The colour components are distributed over the chrominance components **Cb** and Cr. This means that the luminance can be discarded without losing the actual color. The transformation from **RGB** to **YCbCr** is given by

$$Y = 0.299R + 0.587G + 0.11B$$
$$Cb = B - Y$$
$$Cr = R - Y$$

The transformation above is simple and does not require much computation. The fact that luminance can be easily discarded in **YCbCr** and the simple transformation from **RGB** to **YCbCr** makes it a convenient colour space for the skin detection task. All the sample images that are loaded from the dataset are in RGB, they are loaded and then converted to $YCbCr$. Once a sample image is loaded and converted to the relevant colour space its Y component is discarded and only the Cb and Cr are kept. See the functions below that are implemented in source code.

```
def loadSkinImages(path):
    i=0
    D=np.zeros((3354*35*35,2))
    for filename in glob.glob(path+'/*.jpg'):
        im=plt.imread(filename)
        im=color.rgb2ycbcr(im)
        D[i:i+35*35,0]=im[:,:,1].flatten()
        D[i:i+35*35,1]=im[:,:,2].flatten()
        i=i+35*35
    return D

def loadNotSkinImages(path):
    i=0
    D=np.zeros((5590*35*35,2))
    for filename in glob.glob(path+'/*.jpg'):
        im=plt.imread(filename)
        im=color.rgb2ycbcr(im)
        D[i:i+35*35,0]=im[:,:,1].flatten()
        D[i:i+35*35,1]=im[:,:,2].flatten()
        i=i+35*35
    return D
```

The code snippets shown above loads the human skin samples and the not human skin samples respectively. The skin samples are stored in a 4 108 650×2 matrix while the not skin samples are stored in a 6 847 750×2 matrix. The columns of each matrix correspond to Cb and Cr values from sample images.

3.2 Data processing

Once the datasets have been loaded and stored accordingly we want to be able to access descriptive statistics of the samples. Values we want to conveniently be able to compute is the mean, which would be a mean vector $\boldsymbol{\mu} = (\mu_{Cb}, \mu_{Cr})$, standard deviation $\boldsymbol{\sigma} = (\sigma_{Cb}, \sigma_{Cr})$ and the covariance matrix ($\boldsymbol{C}$). These values can be computed for skin samples and non skin samples by specifing the relevant matrix that has stored data. See the functions below that are implemented in source code.

```
#return mean vector
def getMean(data):
    return np.mean(data,axis=0)

#returns standard dev vector
def getStd(data):
    return np.std(data,axis=0)

#returns covariance matrix
def getCov(data):
    cov=np.cov(data.T)
    return cov
```

In the code snippets above the parameter data will be one of the matrices obtained when loading the sample images from dataset as specified earlier.

3.3 Skin detection schemes

Now that we have the relevant data and in the convenient colour space and the means to process it the actual task of skin detection can be tackled. Three approaches were used in this project thresholding, k-means clustering and a gaussian skin colour model.

3.3.1 Thresholding

To perform skin segmentation on an input image using a thresholding approach we first smooth the input image using a gaussian filter with $\sigma = 6$ and then convert the image to $YCbCr$ colour space. We then use the loaded skin samples to obtain a mean vector $\boldsymbol{\mu} = (\mu_{Cb}, \mu_{Cr})$. We also obtain the standard deviation vector $\boldsymbol{\sigma} = (\sigma_{Cb}, \sigma_{Cr})$. To detect skin regions in the smoothed image we compute the euclidean distance of Cb and Cr components of the image from the

mean vector, that is

$$D(x,y) = \sqrt{(Cb(x,y) - \mu_{Cb})^2 + (Cr(x,y) - \mu_{Cr})^2}$$

We then use a scalar multiple the norm of the standard deviation vector as a threshold value

$$T = \alpha ||\boldsymbol{\sigma}||$$

The binary image is obtained as follows

$$im(x,y) = \begin{cases} 1 & if\ D(x,y) < T \\ 0 & otherwise \end{cases}$$

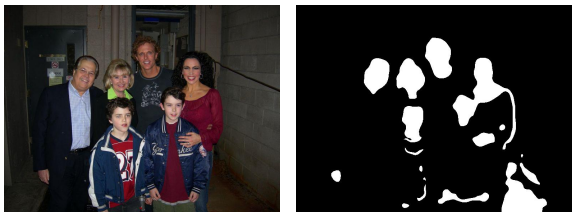
In the implementation in the source we have $\alpha = 1.4$. See the functions below that are implemented in source code.

```
def thresholdSegmentation(image):
    f=image.copy()

    #image smoothing
    f=filters.gaussian(f,sigma=6,multichannel=True)
    #rgb to ycbcr
    ycbcrImage=color.rgb2ycbcr(f)

    #extract relevant channels discard y
    cb=ycbcrImage[:,:,1]
    cr=ycbcrImage[:,:,2]

    #mean vector cb,cr and standard deviation from skin samples stored in X
    mean=getMean(X)
    std=1.4*getStd(X)

    #subtract mean value
    cb=cb-mean[0]
    cr=cr-mean[1]
    d=np.sqrt(cb**2+cr**2)
    t=np.sqrt(np.dot(std,std))

    result=np.zeros(cb.shape)

    #thresholding step
    idx=d<t
    result[idx]=1
    return result
```

The resulting images from using the above method on face images are shown in figure 1 below.

(a) Original image (b) Skin regions image

(c) Original image (d) Skin regions image



(e) Original image (f) Skin regions image

Figure 1: Thresholding method samples

The images in figure 1 above show that the thresholding method performs quite well in detecting skin region on face images. In figure 1(f) the thresholding method was used on a family photo and it achieved decent results although not as accurate as the skin detection in face images.

3.3.2 k-means clustering

The idea is to partition the image into k number of disjoint clusters. Since we want to separate skin regions from non-skin regions we will partition the image into 2 clusters, C_1 skin and C_2 not skin. The initial cluster centers are set to be the mean vectors obtained from the skin samples and not skin samples from dataset. Before any clustering is ran on an input image, the image is smoothed using a gaussian filter with $sigma = 6$ then the image is converted to the $YCbCr$ colour space. Clustering is performed on the image until the residual error E is below a threshold value T ($T = 1$ in this implementation). To obtain a binary image the following is done

$$im(x,y) = \begin{cases} 1 & if\ im(x,y) \in C_1 \\ 0 & if\ im(x,y) \in C_2 \end{cases}$$

See the function below implemented in source code.

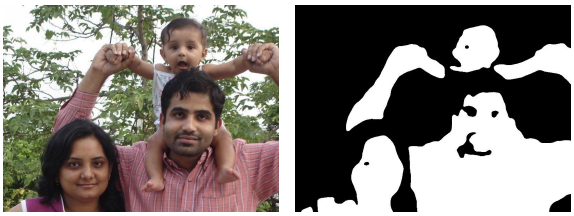
```
def kMeans(image):
    f=image.copy()
    #image smoothing
    f=filters.gaussian(f,sigma=6,multichannel=True)
    #rgb to ycbcr
    ycbcrImage=color.rgb2ycbcr(f)
    shape=image.shape
    c=np.zeros((shape[0],shape[1]))

    #initial cluster centers
    c1=getMean(X)
    c2=getMean(XC)

    while True:
        cb=ycbcrImage[:,:,1]
        cr=ycbcrImage[:,:,2]
        d1=np.sqrt((cb-c1[0])**2+(cr-c1[1])**2)
        d2=np.sqrt((cb-c2[0])**2+(cr-c2[1])**2)

        n1=np.count_nonzero(d1<d2)
        n2=np.count_nonzero(d1>d2)

        idx=d1<d2
        idy=d1>d2

        c[idx]=1
        c[idy]=0

        temp1=c1.copy()
        temp2=c2.copy()
```

```
#cluster updates
c1[0]=np.sum(cb[idx])
c1[1]=np.sum(cr[idx])
c1=c1/n1

c2[0]=np.sum(cb[idy])
c2[1]=np.sum(cr[idy])
c2=c2/n2

#residual error
err=(np.sqrt(np.dot(temp1-c1,temp1-c1)))+(np.sqrt(np.dot(temp2-c2,temp2-
#stopping criteria
if err<1:
    return c
```

The resulting images from using the above method on face images and family photos are shown in figure 2 below.

(a) Original image (b) Skin regions image

(c) Skin regions image (d) Skin regions image



(e) Original image (f) Skin regions image

Figure 2: k-means samples

From the sample images in figure 2 above it is clear that the implemented k means clustering can detect skin regions but it does not perform quite well. It mislabels some regions in the background as skin and minor details such as eyes were missed out (see 2 (d) and (f)).

3.3.3 Gaussian skin color model

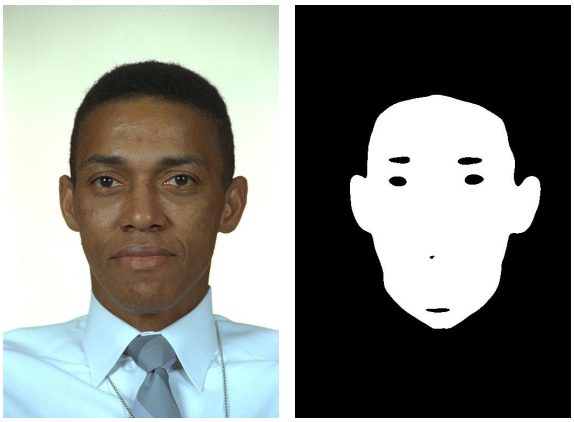
In order to perform skin detection we model the distribution of skin color as a multivariate gaussian. This will allow us to obtain a skin-likelihood image that will indicate the chances of a given pixel being human skin or not. The mean vector and covariance matrix for the multivariate gaussian are obtained from the skin samples. The skin likelihood of a given pixel is given by

$$p(\boldsymbol{x}) = e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T(C^{-1})(\boldsymbol{x}-\boldsymbol{\mu})}$$

where $\boldsymbol{x} = (cb, cr)^T$, $\mu = (\mu_{cb}, \mu_{cr})$ and C is the covariance matrix. The input image is first smoothed using a gaussian filter with $\sigma = 6$, then it is converted to $YCbCr$ colour space and the skin-likelihood image is obtained using the distribution above. The skin-likelihood image is a greyscale image, it is then thresholded using otsu's thresholding method to obtain a binary image outlining skin regions. See implementation below from source code.

```
def gaussianGreyscale(image):
    f=image.copy()
    #image smoothing
    f=filters.gaussian(f,sigma=6,multichannel=True)

    #rgb to ycbcr
    ycbcrImage=color.rgb2ycbcr(f)

    #stuff to compute probaility values
    shape=ycbcrImage.shape
    cb=ycbcrImage[:,:,1].flatten()
    cb=cb.reshape(len(cb),1)
    cr=ycbcrImage[:,:,2].flatten()
    cr=cr.reshape(len(cr),1)
    x=np.hstack((cb,cr))
    mean=getMean(X)
    cov=getCov(X)
    covInverse=np.linalg.inv(cov)
    x=x-mean
    y=x@covInverse
    z=np.sum(x*y,axis=1)
    p=np.exp(-0.5*z).reshape(shape[0],shape[1])

    #thresholding step
    thresh=filters.threshold_otsu(p)
    binary=p>thresh

    return binary
```

The resulting images from using the above method on face images and family photos are shown in figure 3 below.



(a) Original image (b) Skin regions image

(c) Original image (d) Skin regions image

(e) Original image (f) Skin regions image

Figure 3: Gaussian model samples

The samples in figure 3 show that the gaussian skin color model produces adequate results. Although in some instances it fails around the mouth region (see 3 (d)).

4 Results and Analysis

The implemented skin detection schemes have shown to achieve adequate results. We now compare the performance of the 3 algorithms, we use the same input face images to outline the output of each algorithm in figure 4.

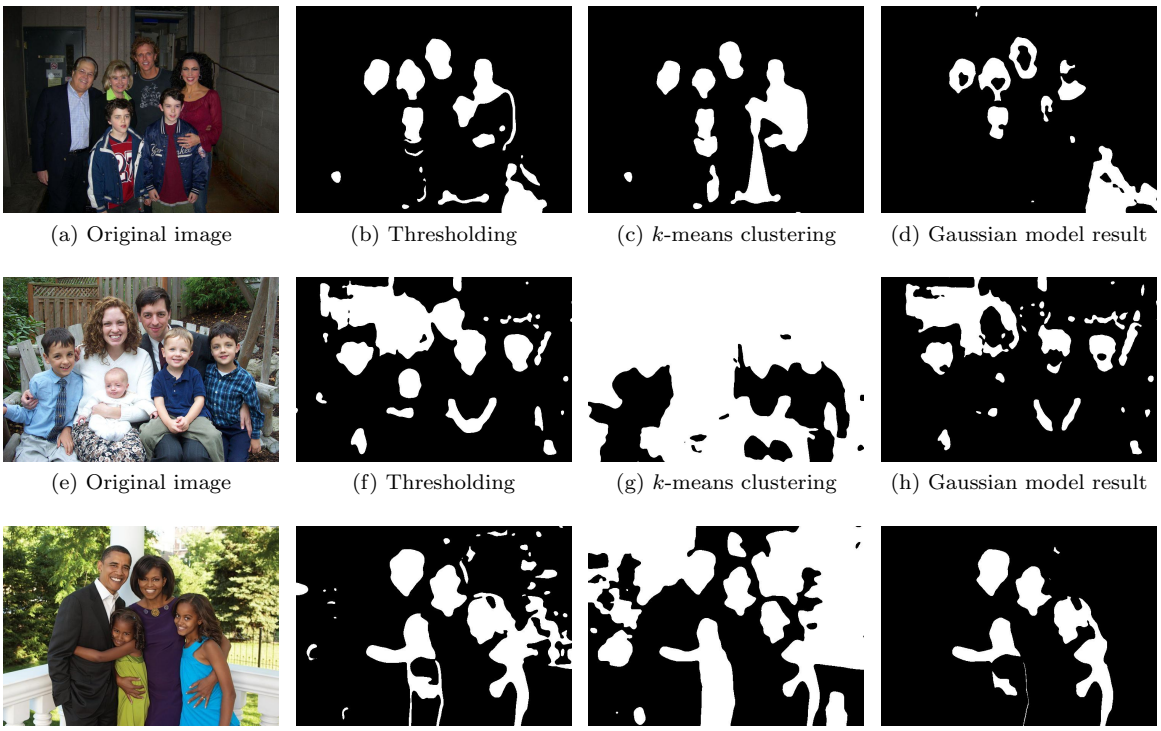
(a) Original image (b) Thresholding (c) k-means clustering (d) Gaussian model result

(e) Original image (f) Thresholding (g) k-means clustering (h) Gaussian model result

(i) Original image (j) Thresholding (k) k-means clustering (l) Gaussian model result

Figure 4: Skin detection schemes comparison on face images

We also compare the performance of the 3 algorithms on family images to outline the output of each algorithm in figure 5. Figure 5 below shows the result of using the different algorithms.



(a) Original image (b) Thresholding (c) k-means clustering (d) Gaussian model result

(e) Original image (f) Thresholding (g) k-means clustering (h) Gaussian model result

(i) Original image (j) Thresholding (k) k-means clustering (l) Gaussian model result

Figure 5: Skin detection schemes comparison on family images

The images in figure 4 and 5 show that the thresholding algorithm provides the best performance for the skin detection task. While k-means clustering outperforms the gaussian skin color model for skin detection in face images it performs poorly on family photos (see figure 5 (c),(g),(k)). In the images in figure 4 and 5 it is important to note that some regions that have been detected as skin are not actually skin, they have been detected as skin because they have colour that is similar skin colour.

The image smoothing process that is performed before any skin detection method is applied helps in improving result. It helps get rid of image noise that would result in sparsely occurring black pixels in the segmented image. Figure 6 below shows the results of gaussian skin color model with and without smoothing.

(a) Original image (b) No smoothing (c) With smoothing

(d) Original image (e) No smoothing (f) With smoothing

Figure 6: Effects of smoothing on resulting image

The image smoothing process has a similar effect in both the thresholding algorithm and the k-means clustering method.

5 Conclusion

The implemented skin detection schemes have yielded good results particularly in the case of face images. The thresholding algorithm can be considered to outperform the k-means clustering and gaussian skin color model. Although the implemented skin detection methods work well with the face images they do not work as well with the family photos. This can be accounted for by the fact that the parameters used in the algorithms (mean vector, standard deviations and covariance matrix) were extracted from skin and not skin samples that were adapted from the face images dataset, which means these parameters may be biased towards face images.

6 Generating results

All the input face images used in figures 1-6 can be found in the *data/face-original* folder while the family input images can be found in *data/family-original* folder. The skin and not skin samples used for parameters can be found in *data/skin-data* and *data/not-skin-data* respectively. The accompanying source code is a jupyter notebook named *main.ipynb* found in *src* folder which will display and save images when run. The *saveImages* function in the notebook will generate segmented face images and family images using the thresholding approach. The output images will be saved in *data/face-segmented* and *data/family-segmented* folders respectively.

7 References

1. [Thabet et al. 2017] Eman Thabet, Fatimah Khalid, Puteri Suhaiza Sulaiman, andRazali Yaakob. Low cost skin segmentation scheme in videos using two alter-native methods for dynamic hand gesture detection method.Advances in Multi-media, 2017, 2017.

2. [Rahman et al. 2013] Md Hafizur Rahman, Tonmoy Das, and Manamatha Sarnaker.Face detection and sex identification from color images using adaboost withsvm based component classifier.International Journal of Computer Applications,76(3):1–6, 2013.

3. [Saxen and Al-Hamadi 2014] Frerk Saxen and Ayoub Al-Hamadi. Color-based skinsegmentation: An evaluation of the state of the art. In2014 IEEE InternationalConference on Image Processing (ICIP), pages 4467–4471. IEEE, 2014.