JUPYTER NOTEBOOK FOR HOMEWORK 2

Segment the foreground from one image and transfer it to another image

In [1]:

```
import numpy as np
import skimage
from skimage.transform import resize
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
import scipy
from scipy.ndimage.filters import convolve
from sklearn.cluster import KMeans

from makeLMfilters import makeLMfilters
```

In [2]:

```
#Convert the image to grayscale


#   Define a function segmentImg which takes in an image and returns the fore
#   The arguments to segmentImg are imgFilename
#   and k (the number of clusters)
def segmentImg(imgFilename, k):
    #1. Load and display the image from which you want to segment the foregro
    #  Make sure to convert your image to grayscale after loading

    image = plt.imread('gecko.jpg')
    print(image.shape)
    #plt.imshow(image)

    gray = rgb2gray(image)
    #plt.imshow(gray, cmap='gray')

    #2. Create an overcomplete bank of filters F (make sure you check the dim
    #
    F = makeLMfilters()
    print(F.shape)

    #3. Convolve the input image with every filter in the bank of filters
    #   to get a response array
    #
    test = np.zeros((gray.shape[0],gray.shape[1],F.shape[2]))
    for i in range(48):
        test[:,:,i] = scipy.ndimage.convolve(gray, F[:,:,i])
    #print(test)

    #4.Take the absolute values of the responses and
    #  reshape the reponse tensor into a matrix of size [row*cols, num_filter
    response = np.absolute(test)
    #print(response)
    X = np.reshape(response, (gray.shape[0]*gray.shape[1], F.shape[2]))
    print(X.shape)
    #plt.imshow(reshaped_img)

    #5. Run k-means on the vectorized reponses X to get a vector of labels (t
    #
    kmeans = KMeans(k, random_state=0)
    kmeans.fit(X)
    k_label = kmeans.labels_
    print(k_label)

    #6. Reshape the label results of k-means so that it has the same size as
    #   Return the label image which we call idx
    idx = np.reshape(k_label, (gray.shape[0], gray.shape[1]))


    print('***Segmentation done***')
    return idx.astype(np.float)
```

In [3]:

```
#segmentImg('dog.jpg', 5)
```

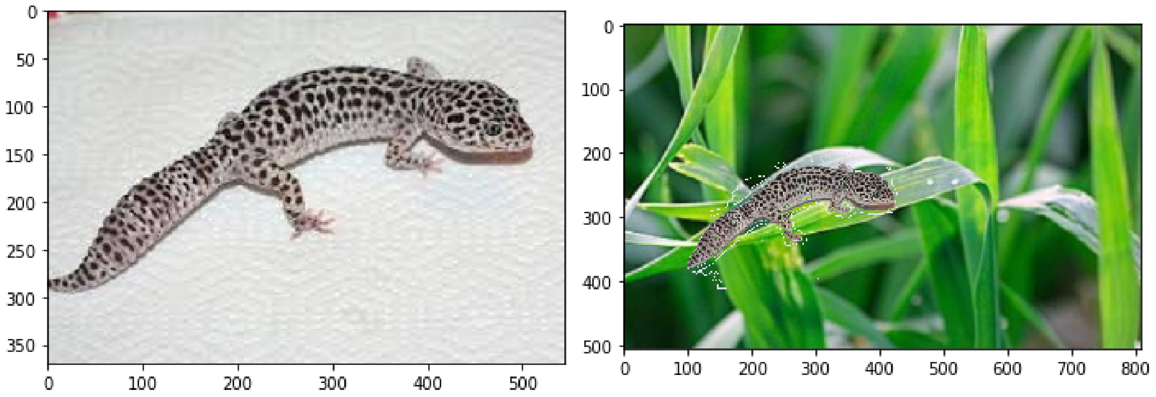
In [4]:

```
"""
You are now given a second function tansferImg with the arguments:
    fgs: foreground label ids from clustering
    idxImg: the reshaped indexed image also from clustering
    sImgFilename: source image (a color image to pick pixels from)
    tImgFilename: target image (also a color image to transfer pixels to)
    This function is customized to work for the images provided for the homew
"""
def transferImg(fgs, idxImg, sImgFilename, tImgFilename):

    # Read the images, estimate their dimensions
    sImg = skimage.io.imread(sImgFilename)
    tImg = skimage.io.imread(tImgFilename)
    rows, cols, clrs = sImg.shape

    # Crop the source and indexed images
    idxImg = idxImg[25:rows-25, 25:cols-25]
    sImg = sImg[25:rows-25, 25:cols-25]
    rows, cols, clrs = sImg.shape

    # Resize them by so they fit in target file, tImg
    idxImg = resize(idxImg,(int(idxImg.shape[0]/1.5), int(idxImg.shape[1]/1.5
    idxImg = idxImg.astype(np.uint8)
    sImg = 255*resize(sImg, (int(sImg.shape[0]/1.5), int(sImg.shape[1]/1.5)),
    sImg = sImg.astype(np.uint8)
    rows, cols, clrs = sImg.shape

    # Transfer idx onto tImg
    for i in range(rows):
        for j in range(cols):
            if idxImg[i,j] in fgs:
                # Coordinate offsets from boundary
                iOff = i+200
                jOff = j+100
                tImg[iOff,jOff,0] = sImg[i,j,0]
                tImg[iOff,jOff,1] = sImg[i,j,1]
                tImg[iOff,jOff,2] = sImg[i,j,2]

    print('***transfer done')
    #plt.imshow(tImg)
    #plt.show()
    return tImg
```

In [5]:

```
#7. Test your segmentation function with an image file and # of clusters, k
# Below is an example of how to call the transfer function
#
image = plt.imread('gecko.jpg')
idx = segmentImg('gecko.jpg', 6)
# Insert only the numbers that correspond to the foreground labels in the tra
trImg = transferImg([0,2,4,5], idx, 'gecko.jpg', 'bg2.jpg')
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
ax = axes.ravel()
ax[0].imshow(image)
ax[1].imshow(trImg)
fig.tight_layout()
plt.show()
```

```
(370, 544, 3)
(49, 49, 48)
(201280, 48)
[0 0 0 ... 1 1 1]
***Segmentation done***
***transfer done
```



In [6]:

```
#8. Write a new function segmentImgClr with the same arguments as segmentImg
#   But this time, instead of features based on filter responses, just use c
#   Try different colorspaces
"""def segmentImgClr(imgFilename, k):
    image = plt.imread('giraffe.jpg')
    print(image.shape)
    #plt.imshow(image)

    #lab = skimage.color.rgb2lab(image)
    lab = skimage.color.rgb2hsv(image)

    #plt.imshow(lab)
    #plt.show()
    print(lab.shape)

    X = np.reshape(lab, (lab.shape[0]*lab.shape[1], lab.shape[2]))
    print(X.shape)

    kmeans = KMeans(k)
    kmeans.fit(X)
    k_label = kmeans.labels_
    print(k_label)

    idx = np.reshape(k_label, (lab.shape[0], lab.shape[1]))

    print('***Segmentation done***')
    return idx.astype(np.float)"""
```

Out[6]:

```
"def segmentImgClr(imgFilename, k):\n    image = plt.imread('giraffe.jpg')
\n    print(image.shape)\n    #plt.imshow(image)\n    \n    #lab = skimage.
color.rgb2lab(image)\n    lab = skimage.color.rgb2hsv(image)\n\n    #plt.im
show(lab)\n    #plt.show()\n    print(lab.shape)\n    \n    X = np.reshape
(lab, (lab.shape[0]*lab.shape[1], lab.shape[2]))\n    print(X.shape)\n
\n    kmeans = KMeans(k)\n    kmeans.fit(X)\n    k_label = kmeans.labels_\n
print(k_label)\n    \n    idx = np.reshape(k_label, (lab.shape[0], lab.shap
e[1]))\n\n    print('***Segmentation done***')\n    return idx.astype(np.fl
oat)"
```

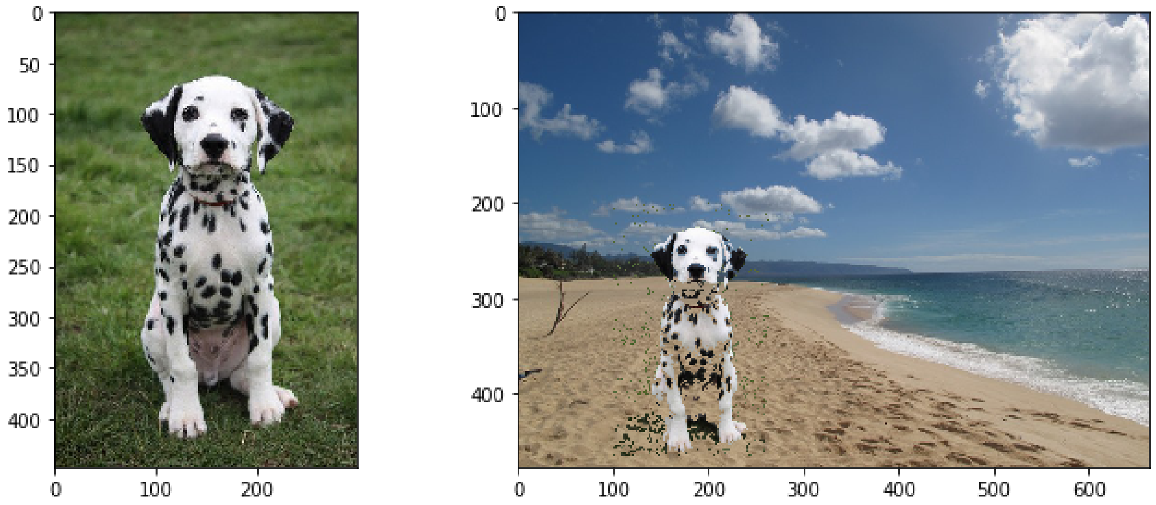
In [7]:

```
#8. Write a new function segmentImgClr with the same arguments as segmentImg
#   But this time, instead of features based on filter responses, just use co
#   Try different colorspaces
def segmentImgClrLab(imgFilename, k):
    image = plt.imread('dog.jpg')
    print(image.shape)
    #plt.imshow(image)

    lab = skimage.color.rgb2lab(image)
    #lab = skimage.color.rgb2hsv(image)

    #plt.imshow(lab)
    #plt.show()
    print(lab.shape)

    X = np.reshape(lab, (lab.shape[0]*lab.shape[1], lab.shape[2]))
    print(X.shape)

    kmeans = KMeans(k, random_state=0)
    kmeans.fit(X)
    k_label = kmeans.labels_
    print(k_label)

    idx = np.reshape(k_label, (lab.shape[0], lab.shape[1]))

    print('***Segmentation done***')
    return idx.astype(np.float)
```

In [22]:

```
#10 Test with a different colorspace
image = plt.imread('dog.jpg')
idx = segmentImgClrLab('dog.jpg', 6)
trImg = transferImg([0,2], idx, 'dog.jpg', 'bg.jpg')#with RGB
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
ax = axes.ravel()
ax[0].imshow(image)
ax[1].imshow(trImg)
fig.tight_layout()
plt.show()
```

```
(448, 298, 3)
(448, 298, 3)
(133504, 3)
[3 3 3 ... 0 0 0]
***Segmentation done***
***transfer done
```



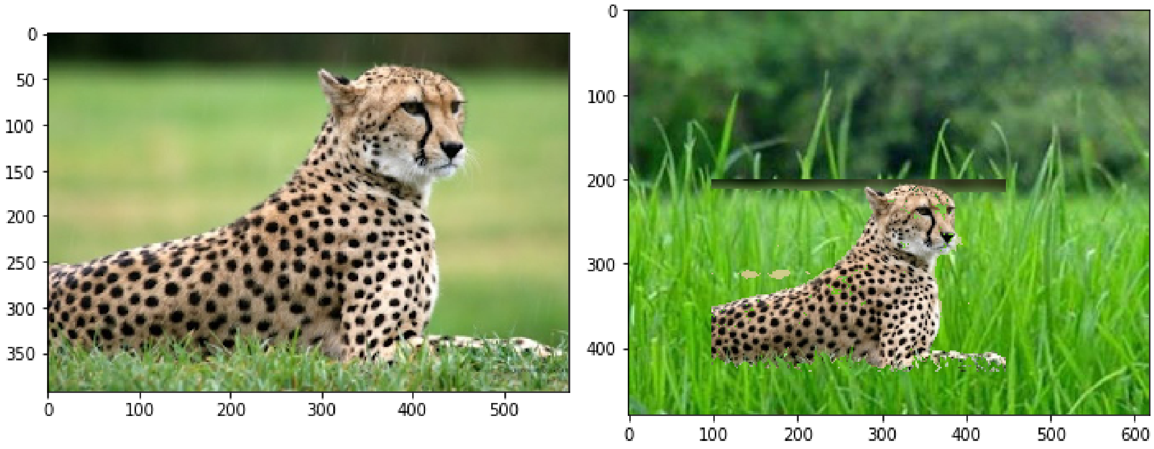
In [9]:

```
#8. Write a new function segmentImgClr with the same arguments as segmentImg
#   But this time, instead of features based on filter responses, just use co
#   Try different colorspaces
def segmentImgClrHsv(imgFilename, k):
    image = plt.imread('cheetah.jpg')
    print(image.shape)
    #plt.imshow(image)

    lab = skimage.color.rgb2hsv(image)

    #plt.imshow(lab)
    #plt.show()
    print(lab.shape)

    X = np.reshape(lab, (lab.shape[0]*lab.shape[1], lab.shape[2]))
    print(X.shape)

    kmeans = KMeans(k, random_state=0)
    kmeans.fit(X)
    k_label = kmeans.labels_
    print(k_label)

    idx = np.reshape(k_label, (lab.shape[0], lab.shape[1]))

    print('***Segmentation done***')
    return idx.astype(np.float)
```

In [10]:

```
#10 Test with a different colorspace
image = plt.imread('cheetah.jpg')
idx = segmentImgClrHsv('cheetah.jpg', 5)
trImg = transferImg([1,2,3,4], idx, 'cheetah.jpg', 'bg3.jpg')#with HSV
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
ax = axes.ravel()
ax[0].imshow(image)
ax[1].imshow(trImg)
fig.tight_layout()
plt.show()
```

```
(392, 571, 3)
(392, 571, 3)
(223832, 3)
[2 2 2 ... 4 0 0]
***Segmentation done***
***transfer done
```



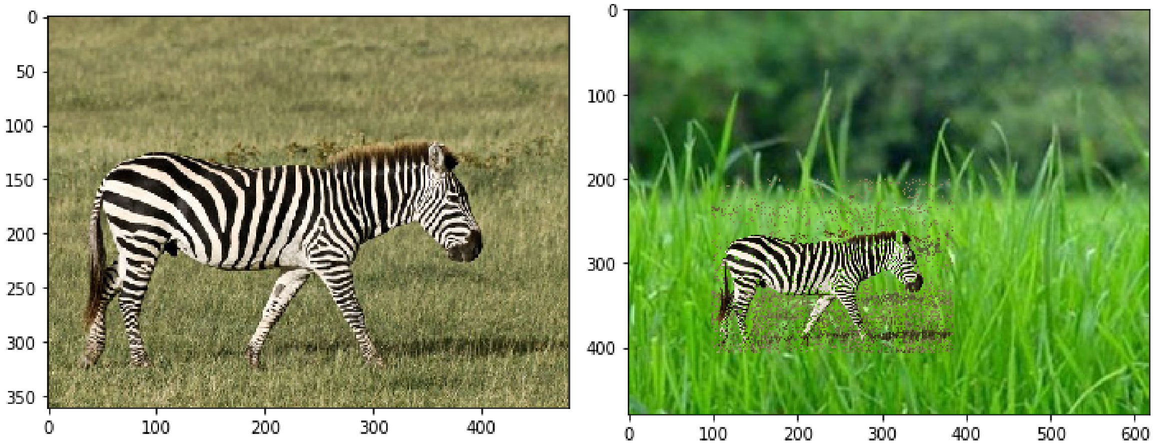
In [11]:

```
#8. Write a new function segmentImgClr with the same arguments as segmentImg
#   But this time, instead of features based on filter responses, just use co
#   Try different colorspaces
def segmentImgClrRgb(imgFilename, k):
    image = plt.imread('zebra.jpg')
    print(image.shape)
    #plt.imshow(image)

    #lab = skimage.color.rgb2hsv(image)

    #plt.imshow(lab)
    #plt.show()
    #print(lab.shape)

    X = np.reshape(image, (image.shape[0]*image.shape[1], image.shape[2]))
    #print(X.shape)

    kmeans = KMeans(k, random_state=0)
    kmeans.fit(X)
    k_label = kmeans.labels_
    print(k_label)

    idx = np.reshape(k_label, (image.shape[0], image.shape[1]))

    print('***Segmentation done***')
    return idx.astype(np.float)
```

In [12]:

```
#10 Test with a different colorspace
image = plt.imread('zebra.jpg')
idx = segmentImgClrRgb('zebra.jpg', 6)
trImg = transferImg([1,2], idx, 'zebra.jpg', 'bg3.jpg')#with RGB
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
ax = axes.ravel()
ax[0].imshow(image)
ax[1].imshow(trImg)
fig.tight_layout()
plt.show()
```

```
(360, 480, 3)
[0 0 5 ... 5 5 4]
***Segmentation done***
***transfer done
```



In [13]:

```
#9. Again test your new color-based segmentation function
#   The same transferImg function should work here also.
"""image = plt.imread('dog.jpg')
idx = segmentImgClrLab('dog.jpg', 6)
trImg = transferImg([0,2], idx, 'dog.jpg', 'bg.jpg')#with LAB
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
ax = axes.ravel()
ax[0].imshow(image)
ax[1].imshow(trImg)
fig.tight_layout()
plt.show()"""
```

Out[13]:

```
"image = plt.imread('dog.jpg')\nidx = segmentImgClrLab('dog.jpg', 6)\ntrImg
= transferImg([0,2], idx, 'dog.jpg', 'bg.jpg')#with LAB\nfig, axes = plt.su
bplots(1, 2, figsize=(10, 4))\nax = axes.ravel()\nax[0].imshow(image)\nax
[1].imshow(trImg)\nfig.tight_layout()\nplt.show()"
```

In [14]:

```
#10 Test with a different colorspace
"""image = plt.imread('cheetah.jpg')
idx = segmentImgClrHsv('cheetah.jpg', 5)
trImg = transferImg([1,2,3,4], idx, 'cheetah.jpg', 'bg3.jpg')#with HSV
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
ax = axes.ravel()
ax[0].imshow(image)
ax[1].imshow(trImg)
fig.tight_layout()
plt.show()"""
```

Out[14]:

```
"image = plt.imread('cheetah.jpg')\nidx = segmentImgClrHsv('cheetah.jpg',
5)\ntrImg = transferImg([1,2,3,4], idx, 'cheetah.jpg', 'bg3.jpg')#with HSV
\nfig, axes = plt.subplots(1, 2, figsize=(10, 4))\nax = axes.ravel()\nax
[0].imshow(image)\nax[1].imshow(trImg)\nfig.tight_layout()\nplt.show()"
```

In [15]:

```
#10 Test with a different colorspace
"""image = plt.imread('zebra.jpg')
idx = segmentImgClrRgb('zebra.jpg', 6)
trImg = transferImg([1,2], idx, 'zebra.jpg', 'bg3.jpg')#with RGB
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
ax = axes.ravel()
ax[0].imshow(image)
ax[1].imshow(trImg)
fig.tight_layout()
plt.show()"""
```

Out[15]:

```
"image = plt.imread('zebra.jpg')\nidx = segmentImgClrRgb('zebra.jpg', 6)\nt
rImg = transferImg([1,2], idx, 'zebra.jpg', 'bg3.jpg')#with RGB\nfig, axes
= plt.subplots(1, 2, figsize=(10, 4))\nax = axes.ravel()\nax[0].imshow(imag
e)\nax[1].imshow(trImg)\nfig.tight_layout()\nplt.show()"
```