AULA PRÁTICA 7. *PRACTICAL CLASS 7.*

1. CRESCIMENTO DE REGIÕES. *REGION GROWING.*

```
import numpy as np
import matplotlib.pyplot as plt
from imageio import imread
from scipy import ndimage
from skimage.morphology import rectangle, erosion, dilation, binary_erosion, \
binary_dilation, watershed
from copy import deepcopy
```

1.1 Determinar os mínimos regionais do gradiente morfológico da imagem **rgrow.tif**, e determinar a correspondente imagem de *watershed*. *Determine the regional minima of the morphological gradient of the **rgrow.tif** image, and determine the corresponding image of watershed.*

1.2 Eliminar as bacias de *watershed* que tocam o bordo da janela. *Eliminate watershed basins that touch the edge of the window.*

1.3 A partir do pixel-semente (8, 15), reconstruir a correspondente bacia de *watershed*. *From the seed pixel (8, 15), reconstruct the corresponding watershed basin.*

```
seed = [[8, 15]]
B = W==W[seed[0][0], seed[0][1]]
```

1.4 Por anexação de bacias obter a região que satisfaça o seguinte critério: a diferença absoluta entre a média dos pixels da região obtida por anexação, e a média dos pixels da bacia a anexar deve ser inferior a 10. *By basin attachment obtain the region that meets the following criteria: the absolute difference between the pixel average of the region obtained by attachment and the pixel average of the basin to be attached must be less than 10.*

```
m = [-2, -1]
while m[-1]!=m[-2]:
    A = B*F.astype(float)
    m.append(np.mean(A[np.nonzero(A)]))
    D = np.logical_and(binary_dilation(B, ee), np.logical_not(B))*W
    val = np.unique(D[np.nonzero(D)])
    for j in range(len(val)):
        V = (D==val[j])*F.astype(float)
        if abs(np.mean(V[np.nonzero(V)])-np.mean(m[2:len(m)]))<10:
            B = np.logical_or(B, W==val[j])

del m[0:2]; del m[-1]
```

2. OPERADOR "CANNY EDGE DETECTOR". *"CANNY EDGE DETECTOR" OPERATOR.*

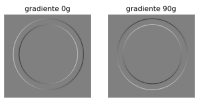
```
import numpy as np
import matplotlib.pyplot as plt
from imageio import imread
from scipy import ndimage
from skimage.morphology import rectangle, binary_dilation
from copy import deepcopy
```

2.1 Filtrar a imagem **circle.tif**, com um filtro passa-baixa Gaussiano. *Filter the **circle.tif** image with a Gaussian low pass filter.*

```
gauss_denoised = ndimage.filters.gaussian_filter(F.astype(float), 1)
```

2.2 Aplicar o operador bi-direccional de Sobel. *Apply the bi-directional Sobel operator.*

```
sx = np.array([[ ?,  0,  ?],
               [ ?,  0,  ?],
               [ ?,  0,  ?]], dtype='float')
sy = np.array([[ ?,  ?,  ?],
               [ 0,  0,  0],
               [ ?,  ?,  ?]], dtype='float')
conv000 = ndimage.convolve(gauss_denoised.astype(float), sx, mode='constant', cval=0.0)
conv090 = ndimage.convolve(gauss_denoised.astype(float), sy, mode='constant', cval=0.0)
```



2.3 Determinar a imagem correspondente à inclinação do gradiente em cada pixel. *Determine the image corresponding to the slope of the gradient in each pixel.*

```
z = np.logical_and(conv090==0, conv000==0)*1
D0 = np.arctan2(conv090, conv000) # angulo -pi:pi
D1 = D0*180/np.pi # angulo -180:180
Theta = D1+180 # angulo 0:360
```

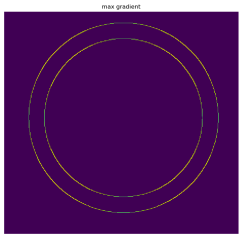
2.4 Quantização das inclinações para valores de 45°, 90°, 135° e 180°. *Quantization of slopes for values of 45°, 90°, 135° and 180°.*

```
Theta0 = np.logical_or(np.logical_and(Theta>=0, Theta<22.5), \
                       np.logical_and(Theta>=337.5, Theta<=360))
Theta45 = np.logical_and(Theta>=22.5, Theta<67.5)
Theta90 = np.logical_and(Theta>=67.5, Theta<112.5)
Theta135 = np.logical_and(Theta>=112.5, Theta<=157.5)
Theta180 = np.logical_and(Theta>=157.5, Theta<202.5)
Theta225 = np.logical_and(Theta>=202.5, Theta<247.5)
Theta270 = np.logical_and(Theta>=247.5, Theta<292.5)
Theta315 = np.logical_and(Theta>=292.5, Theta<337.5)

Tt1 = np.logical_or(Theta45, Theta225)*45
Tt2 = np.logical_or(Theta90, Theta270)*90
Tt3 = np.logical_or(Theta135, Theta315)*135
Tt4 = np.logical_or(Theta180, Theta0)*180
ThetaQuantized = (Tt1+Tt2+Tt3+Tt4).astype(float)
ThetaQuantized[z==1] = 0
```

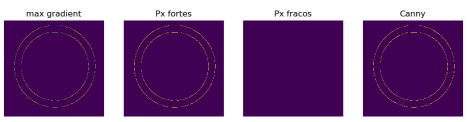
2.5 Determinação dos pixels máximos do gradiente, para os valores quantizados. *Determination of the maximum gradient pixels for the quantized values.*

```
S = (abs(conv000)+abs(conv090))
G1 = np.zeros((lin, col))
for k1 in range(1, lin-1):
    for k2 in range(1, col-1):
        if (k1==0 | k1==lin-1 | k2==0 | k2==col-1):
            G1[k1, k2] = 0
        else:
            if ThetaQuantized[k1, k2]==180:
                if np.logical_and(S[k1, k2-1]<S[k1, k2], S[k1, k2+1]<S[k1, k2]):
                    G1[k1, k2] = S[k1, k2]
            if ThetaQuantized[k1, k2]==45:
                if np.logical_and(S[k1+1, k2-1]<S[k1, k2], S[k1-1, k2+1]<S[k1, k2]):
                    G1[k1, k2] = S[k1, k2]
            if ThetaQuantized[k1, k2]==90:
                if np.logical_and(S[k1-1, k2]<S[k1, k2], S[k1+1, k2]<S[k1, k2]):
                    G1[k1, k2] = S[k1, k2]
            if ThetaQuantized[k1, k2]==135:
                if np.logical_and(S[k1-1, k2-1]<S[k1, k2], S[k1+1, k2+1]<S[k1, k2]):
                    G1[k1, k2] = S[k1, k2]
```



2.6 Determinar as linhas de fronteira. Para o efeito, aplicar à imagem resultante anterior um duplo *threshold*, classificando os pixels, como "Fortes" e "Fracos", e anexar, por *histeresis*, os pixels "Fracos" elegíveis, aos pixels "Fortes". *Determine the edge lines. To do this, apply a double threshold to the previous resultant image, classifying the pixels as "Strong" and "Weak", and attach, by hysteresis, the "Weak" eligible pixels to the "Strong" ones.*

```
t1 = ?
t2 = ?
tsup = t2*np.max(G1)/255
tinf = t1*np.max(G1)/255
PxFortes = G1>=tsup
PxFracos = np.logical_and(G1>=tinf, G1<tsup)
Edges = deepcopy(PxFortes)
_, n = ndimage.label(Edges)
a = -1;
while a!=n:
    _, n = ndimage.label(Edges)
    se = rectangle(3, 3)
    Edges = np.logical_or(Edges, np.logical_and(binary_dilation(Edges, se), PxFracos))
    _, a = ndimage.label(Edges)
```



2.7 Aplicar o algoritmo à imagem **lsat01.tif**. *Apply the algorithm to the image **lsat01.tif**.*