

Técnicas Clássicas de Reconhecimento de Padrões (2020/01)

Exercício 01

Ramon Gomes Durães de Oliveira

O exercício consiste na detecção de bordas em imagens aéreas da floresta amazônica utilizando filtros de borda binários num formato de redes neurais sem pesos, ou seja, simulando uma memória RAM. Em resumo, o trabalho foi conduzido da seguinte forma:

1. Amostragem visual de um pixel verde e um marrom
2. Binarização das imagens de acordo com a distância euclidiana dos pixels com os pixels verde e marrom amostrados.
3. Detecção das bordas comparando a imagem binária com uma série de filtros de borda 3x3 definidos à mão.

Abaixo está o passo a passo do trabalho. Os resultados são mostrados na última tabela.

Preparação do ambiente

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image
import glob
from scipy.spatial.distance import pdist

## Conversion script
#for image_name in glob.glob(".*\\data\\*.jpg"):
#    split_name = image_name.split("\\")
#    new_name = split_name[0:-1]
#    print("rgb_" + split_name[-1])
#    new_name.append("rgb_" + split_name[-1])
#    print(new_name)
#    img = Image.open(image_name)
#    img.convert('RGB').save(".".join(new_name), "PNG", optimize=True)
```

Amostragem dos Pixels

Pixel verde (a partir da imagem da floresta)

```
In [4]: img = Image.open("..\data\rgb_forest1.jpg")  
img_array = np.array(img)  
img
```

Out[4]:



Escolhendo uma cor e visualizando a cor escolhida:

```
In [5]: green_pixel = img_array[220,220,:]  
green_pixel
```

Out[5]: array([38, 57, 44], dtype=uint8)

```
In [6]: Image.fromarray(np.reshape(np.repeat(green_pixel.reshape(1,1,3), 1600,axis=1),  
(40,40,3)))
```

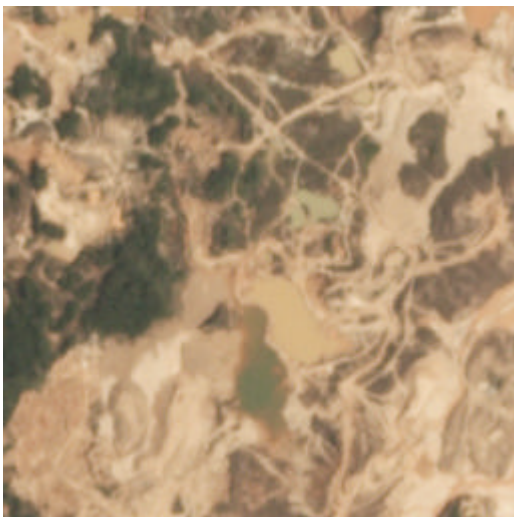
Out[6]:



Pixel marrom (escolhido {a partir de uma imagem com muita terra)

```
In [7]: img = Image.open("..\data\rgb_train_274.jpg")
img_array = np.array(img)
img
```

Out[7]:



Escolhendo uma cor e visualizando a cor escolhida:

```
In [9]: brown_pixel = img_array[255,255,:]
brown_pixel
```

Out[9]: array([231, 205, 171], dtype=uint8)

```
In [10]: Image.fromarray(np.reshape(np.repeat(brown_pixel.reshape(1,1,3), 1600,axis=1),
(40,40,3)))
```

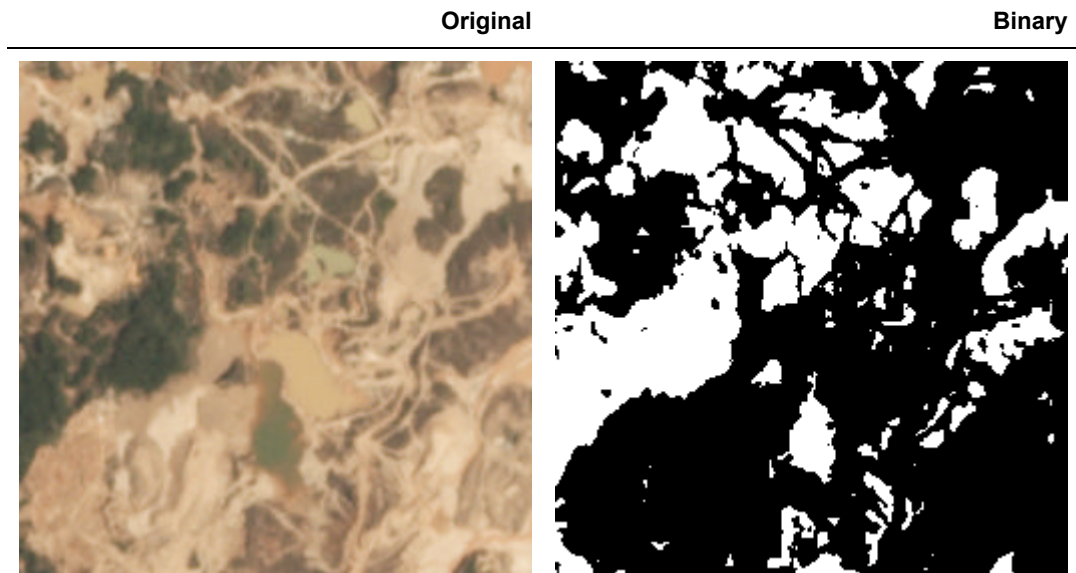
Out[10]:



Gerando as imagens binárias

```
In [12]: ## Binarization script
for image_name in glob.glob("..\data\rgb_*.jpg"):
    split_name = image_name.split("\\")
    new_name = split_name[0:-1]
    new_name.append("bin_"+split_name[-1])
    img = Image.open(image_name)
    img_array = np.array(img)
    img_shape = img_array.shape
    bin_array = np.zeros((img_shape[0],img_shape[1]), dtype='uint8')
    for i in range(img_shape[0]):
        for j in range(img_shape[1]):
            dist_brown = np.abs(pdist((brown_pixel, img_array[i,j,:]))[0])
            dist_green = np.abs(pdist((green_pixel, img_array[i,j,:]))[0])
            bin_array[i,j] = 255 if (dist_green < dist_brown) else 0
    Image.fromarray(bin_array).save("\\".join(new_name), "PNG", optimize=True)
```

Visualizando um exemplo:



Definindo filtros de bordas binários manualmente

```

In [13]: filters = [# verticais
    "100100100",
    "110110110",
    "010010010",
    "001001001",
    "011011011",
    # horizontais
    "111000000",
    "111111000",
    "000111000",
    "000111111",
    "000000111",
    # diagonais horizontais
    "100000000",
    "110000000",
    "111100000",
    "111110000",
    "111111100",
    "111111110",
    "011111111",
    "001111111",
    "000011111",
    "000001111",
    "000000011",
    "000000001",
    # diagonais verticais
    "100100000",
    "110100100",
    "110110100",
    "111110110",
    "111111110",
    "001000000",
    "001001000",
    "011001001",
    "011011001",
    "001011011",
    "001001011",
    "111011011",
    "111111011",
    "011011111",
    # diagonais
    "100010001",
    "100110111",
    "111011001",
    "001010100",
    "001011111",
    "111110100"]


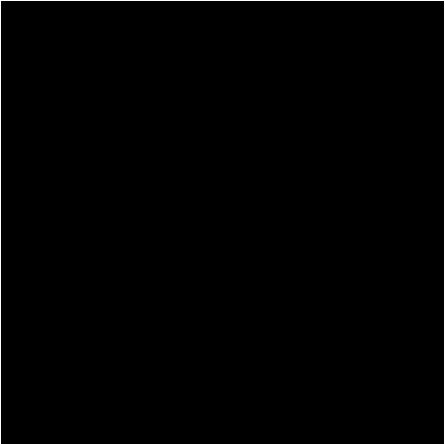

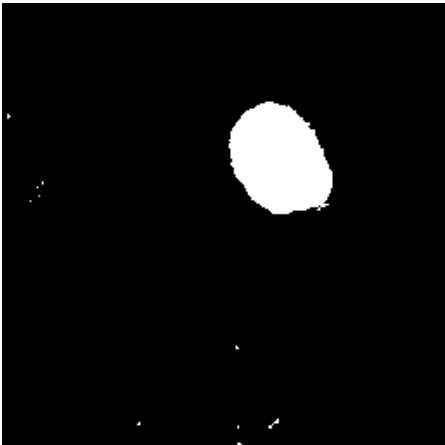
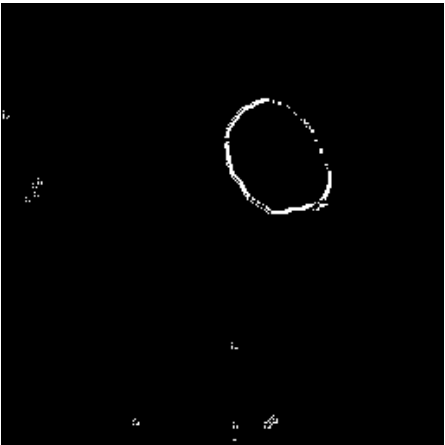


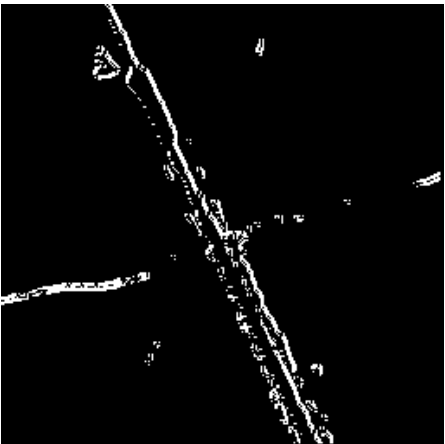
```

Extraindo bordas (janelas cujos "endereço" resultam numa posição de memória com 1)

```
In [14]: for image_name in glob.glob(".\\data\\bin_*.jpg"):
    split_name = image_name.split("\\")
    new_name = split_name[0:-1]
    print("edge_"+split_name[-1])
    new_name.append("edge_"+split_name[-1])
    img = Image.open(image_name)
    img_array = np.array(img)
    img_shape = img_array.shape
    edge_array = np.zeros((img_shape[0],img_shape[1]), dtype='uint8')
    window_size = 3
    for i in range(img_shape[0]-2):
        for j in range(img_shape[1]-2):
            window = img_array[i:i+3, j:j+3]/255
            address = "".join([str(int(x)) for x in window.flatten().tolist
            ()])
            if address in filters:
                edge_array[i,j] = 255
    Image.fromarray(edge_array).save("\\".join(new_name), "PNG", optimize=True
    )
```

```
edge_bin_rgb_forest1.jpg
edge_bin_rgb_train_126.jpg
edge_bin_rgb_train_1591.jpg
edge_bin_rgb_train_1816.jpg
edge_bin_rgb_train_270.jpg
edge_bin_rgb_train_274.jpg
edge_bin_rgb_train_565.jpg
edge_bin_rgb_train_957.jpg
```

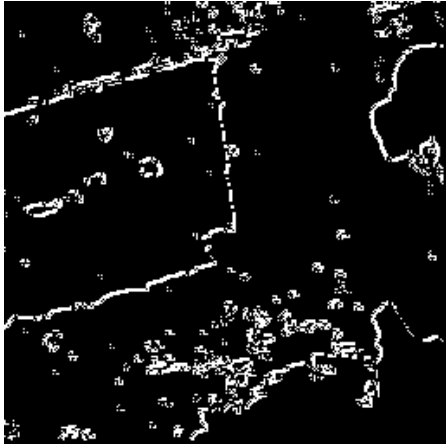
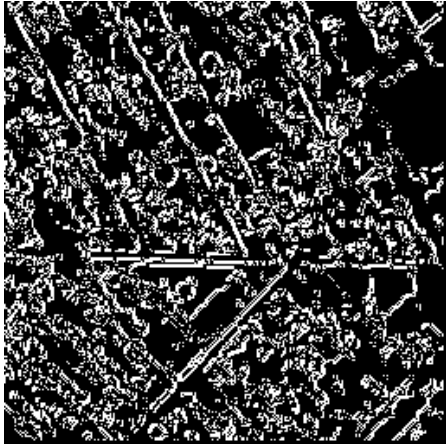
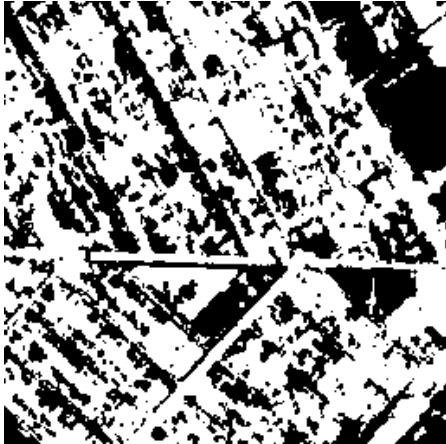
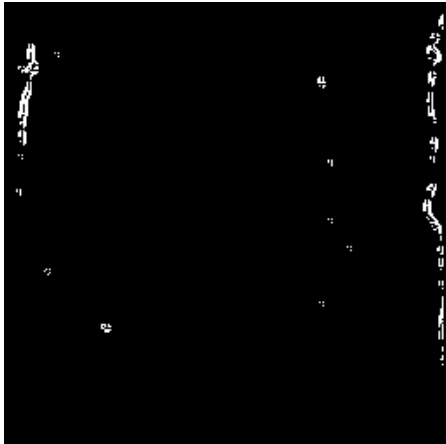
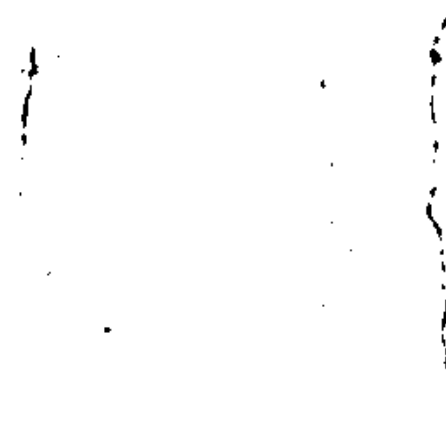
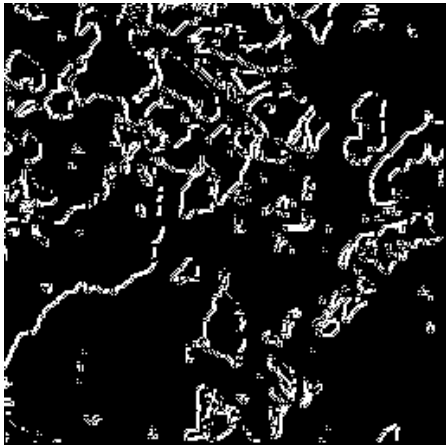
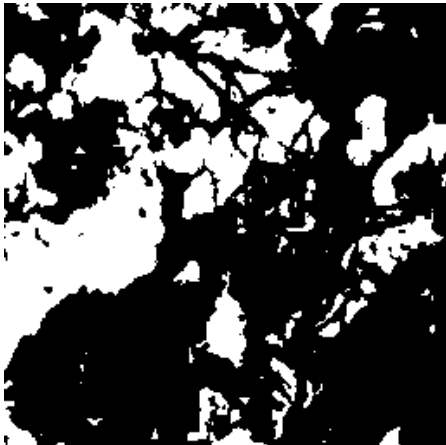
Tabela de Resultados

Original	Binary	Edges
		
		
		

Original

Binary

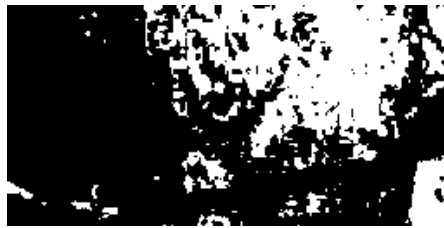
Edges



Original

Binary

Edges



Conclusão

O resultado obtido é interessante, especialmente se levarmos em consideração a simplicidade do método e seu baixo custo computacional e de memória. O resultado pode ainda ser melhorado utilizando técnicas mais maduras de binarização de imagens e filtrando o resultado para que as bordas fiquem mais bem definidas.