

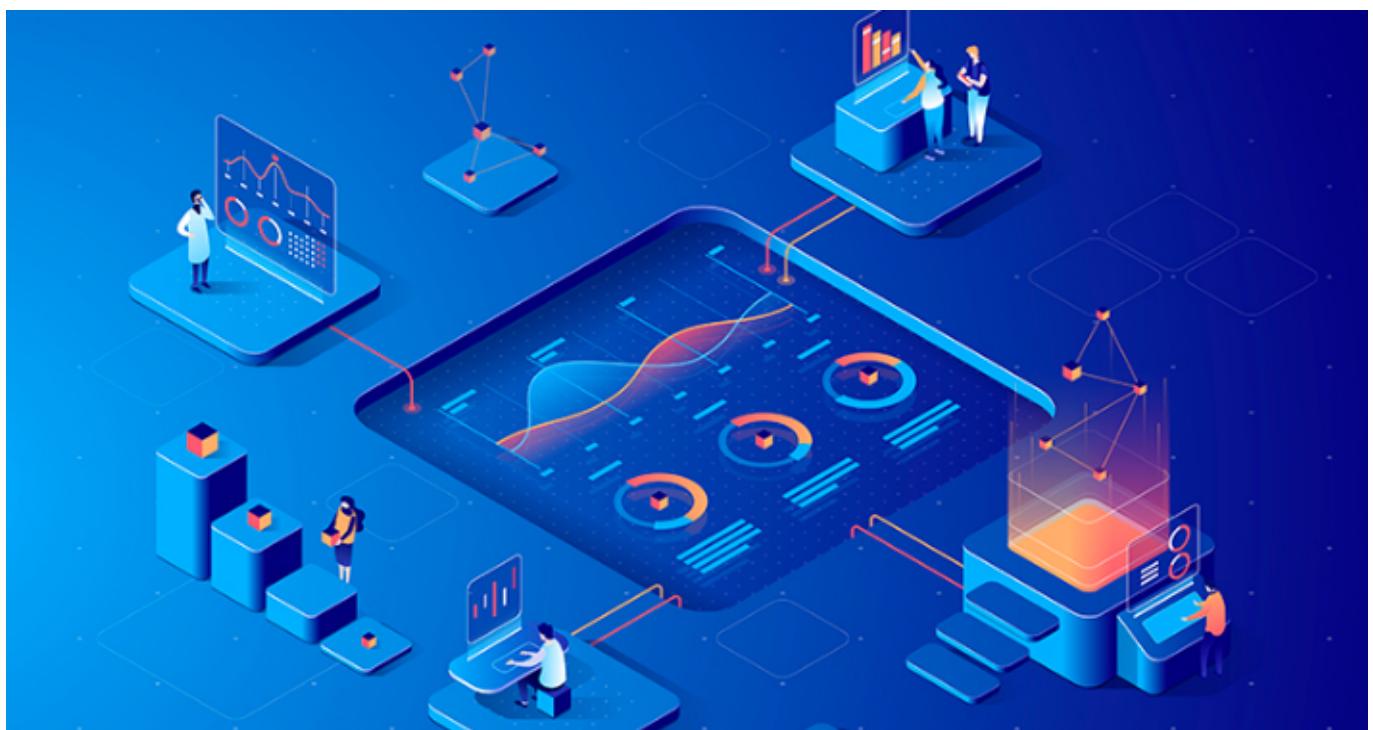
Segmente imagens com Numpy, Skimage, Matplotlib e Imageio



Walter [Follow](#)

Sep 30, 2018 · 6 min read

Extraia objetos de imagens utilizando Python 3 e módulos de processamento de imagens



[Isometric data analysis concept](#)

Introdução

Neste artigo apresentarei técnicas de segmentação de imagens através de uma abordagem prática.

O processo de segmentar imagens tem como objetivo extrair objetos de imagens a partir de diferentes técnicas, combinadas, ou não. Tal conceito é amplamente aplicado em diferentes contextos, sendo um dos mais populares a segmentação em algoritmos de veículos autônomos, como por exemplo a técnica Semantic Segmentation.

Requisitos

Conhecimentos básicos de desenvolvimento em Python 3 e os módulos Numpy, Skimage, Imageio e Matplotlib instalados são requeridos. Você pode realizar o download dos módulos mencionados através do comando:

```
1 pip install scikit-image numpy matplotlib imageio
```

Requirements.sh hosted with ❤ by GitHub

[view raw](#)

Você também pode utilizar o [Colab Research](#) ou a distribuição científica [Anaconda](#)

Imagens

Nos anos 50 a aplicação de Inteligência Artificial como meio de processar imagens era realizada de forma analógica, através da criação manual de regras por programadores. Essa abordagem ficou conhecida como Inteligência Artificial Simbólica.

Atualmente o processamento de imagens pode ser realizado através de Inteligência Artificial e de seus subcampos Aprendizagem de Máquina e Aprendizagem Profunda, de forma digital e automática. Uma abordagem ‘manual’, moderna, será abordada aqui.

A imagem utilizada neste artigo é uma ilustração do grupo de comédia Monty Python:



[Monty Python](#)

Para realizar a abertura da imagem acima utilizaremos o módulo Imageio. O módulo Imageio fornece duas funções I/O : `imread` e `imsave`. A função `imread` recebe como argumento o path da imagem a ser aberta e função `imsave` recebe como argumento o path onde a imagem deve ser salva e um arranjo Numpy.

Abra imagem do Monty Python utilizando a função `imread` passando como argumento o path da imagem:

```
1 # Image I/O
2
3 import imageio
4
5 # Read image
6
7 monty = imageio.imread('monty.png')
```

Open.py hosted with ❤ by GitHub

[view raw](#)

Imagens são compostas por pixels, cada pixel é um valor numérico que representa a intensidade de cor presente em um pixel. Na imagem em questão o valor numérico presente em cada pixel é do tipo `uint8` , que possui um alcance de intensidade de 0–255. A imagem também possui o padrão de cores RGB, isso significa que a imagem possui três canais de cores : red (vermelho), green (verde) e blue (azul).

É necessário que convertamos a imagem do tipo `unit8` para o tipo `float` , pois a cada operação realizada sobre a imagem `unit8` dados são perdidos, contudo ao converter a imagem para `float` , a perda ocorre apenas quando operações de I/O são realizadas. É necessário também que convertamos a imagem do canal de cores RGB para o canal gray (cinza), pois essa operação reduz a complexidade em lidar com a imagem, a quantidade de pixels total é reduzida, pois apenas um canal é utilizado, e possibilita a aplicação de algoritmos que não aceitam imagens RGB.

Para realizar a conversão de `unit8` para `float` , utilizaremos a função `img_as_float` do módulo Skimage. A função `img_as_float` recebe como argumento uma imagem (arranjo Numpy) e converte a mesma para o tipo `float64` , com um alcance de 0-1.

Para converter a imagem de RGB para gray utilizaremos a função `rgb2gray` do módulo Skimage. A função `rgb2gray` recebe como argumento uma imagem e converte a mesma para o canal de cores gray.

Converta a imagem para `float64` e utilize a mesma como input para a função `rgb2gray` :

```
1 # Image processing
2
3 from skimage import color, img_as_float
4
5 # Convert the image to float and gray
6
7 monty_gray = color.rgb2gray(img_as_float(monty))
```

Convert.py hosted with ❤ by GitHub

[view raw](#)

Para verificar o canal e o tipo de dado da imagem original e da imagem após a conversão imprima na tela o `shape` e `dtype` de ambas :

```
1 # Print shape and data type of images
2
3 print(f'\nMonty shape {monty.shape} and type {monty.dtype}\n')
4
5 print(f'Monty gray shape {monty_gray.shape} and type {monty_gray.dtype}', end='\n' * 2)
```

Metadata.py hosted with ❤ by GitHub

[view raw](#)

Para visualizar o resultado do processamento realizado acima vamos utilizar a função `figure`, `imshow` e `show` do módulo Matplotlib. A função `figure` cria um ‘buffer’ que armazena uma image, a função recebe diversos argumentos, porém nesta etapa utilizaremos apenas o `figsize`, que especifica a largura e a altura da imagem em polegadas. A função `imshow` recebe uma imagem e adiciona a mesma ao ‘buffer’. A função `show` mostra a imagem armazenada no ‘buffer’.

Crie uma figura utilizando a função `figure` e adicione a imagem em questão, convertida para `float` e `gray`, a tal figura. Mostre a imagem armazenada no ‘buffer’ utilizando a função `show` (as funções `title` e `axis` são opcionais) :

```
1 # Plots
2
3 import matplotlib.pyplot as pp
4
5 # Plot image monty gray
6
7 pp.figure(figsize=(10, 5))
8
9 pp.imshow(monty_gray, cmap='gray')
10
11 pp.axis('off')
12
13 pp.title('Monty Python')
14
15 pp.show()
```

Figure.py hosted with ❤ by GitHub

[view raw](#)

O resultado da conversão de tipo e remoção de canais :





Plot 1 — Monty Python — Gray e Float

Para que a imagem seja apresentada em preto e branco é necessário que o canal de cores utilizado pelo Matplotlib seja `cmap='gray'`.

Filtros

Filtros possibilitam remover ou adicionar características as imagens. Nesta sessão aplicaremos filtros de ajuste de contraste / brilho e segmentação; para tornar a distribuição de pixels da imagem normalizada e separar os objetos da imagem, convertendo a mesma para binária.

Uma imagem pode possuir valores intensos ou ténues em um certo alcance, tal característica pode indicar se uma imagem possui um bom contraste ou a mesma é escura ou clara. Tal característica pode ser visualizada a partir da plotagem de um histograma. Normalizar a distribuição de pixels pode, ou não, ‘melhorar’ a qualidade de imagens.

Para normalizar a imagem em questão utilizaremos a função `equalize_hist` do módulo Skimage. A função `equalize_hist` recebe uma imagem como argumento e baseando-se em seu histograma realiza o alargamento do contraste.

Utilize a função `equalize_hist` passando como argumento a imagem processada na sessão anterior :

```
1 # Image processing
2
3 from skimage import exposure
4
5 # Histogram equalization
6
```

```
7 monty_hist = exposure.equalize_hist(monty_gray)
```

Hist.py hosted with ❤ by GitHub

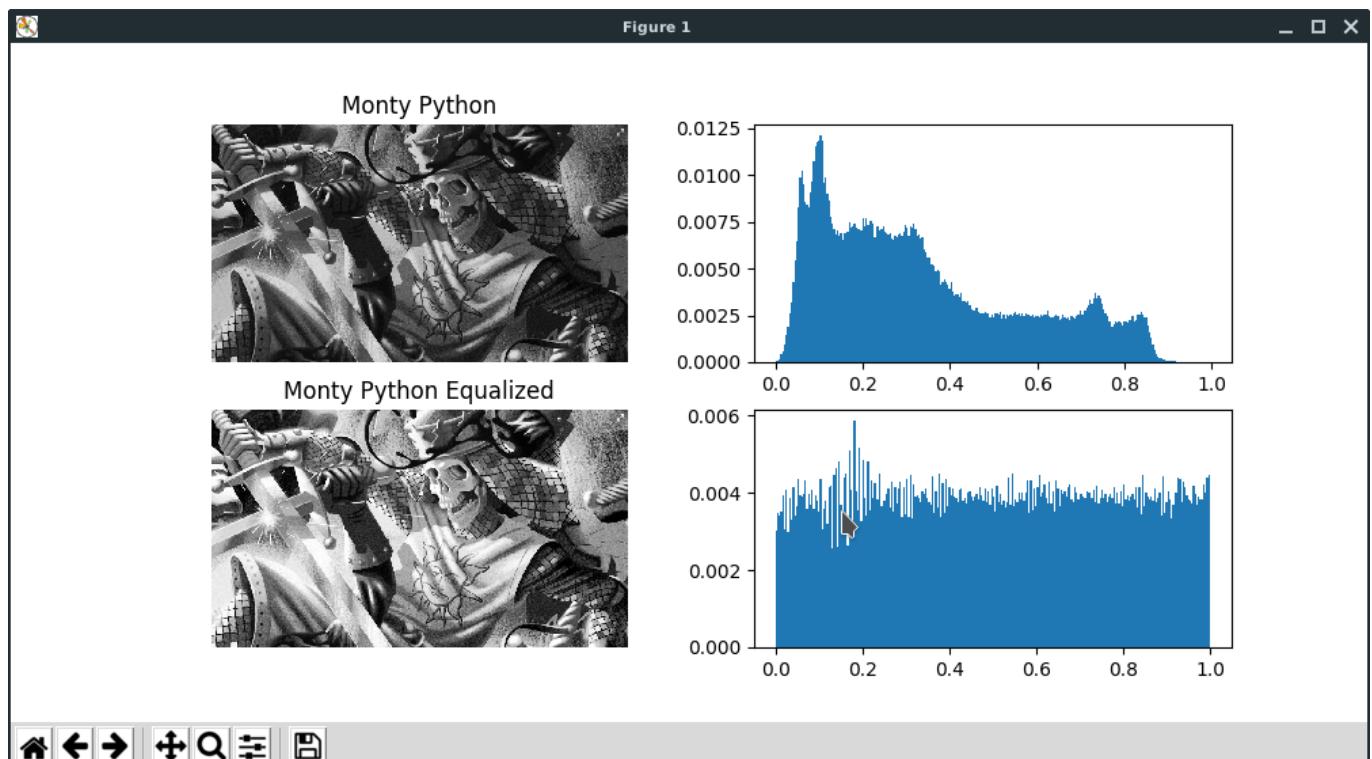
[view raw](#)

O resultado antes / após a normalização :

```
1 # Plot normal and equalized image
2
3 fig, ax = pp.subplots(2, 2, figsize=(10, 5)) # Lines and cols
4
5 cache = [(monty_gray, 'Monty Python'), (monty_hist, 'Monty Python Equalized')]
6
7 for indice, image in zip(range(0, 2), cache):
8     ax[indice, 0].imshow(image[0], cmap='gray')
9
10    ax[indice, 0].set_title(image[1])
11
12    ax[indice, 0].axis('off')
13
14    weights = np.ones(image[0].ravel().shape) / float(image[0].size)
15
16    ax[indice, 1].hist(image[0].flatten(), bins=256, weights=weights)
17
18 pp.show()
```

PlotHist.py hosted with ❤ by GitHub

[view raw](#)



Plot 2 — Monty Python — Histograma

Para segmentar a imagem utilizaremos a função `threshold_otsu` do módulo Skimage. A função `threshold_otsu` recebe como argumento uma imagem e separa os objetos da imagem de forma binária. O retorno da função `threshold_otsu` é um um limiar `float64` que pode ser utilizado como uma máscara na filtragem dos pixels da imagem.

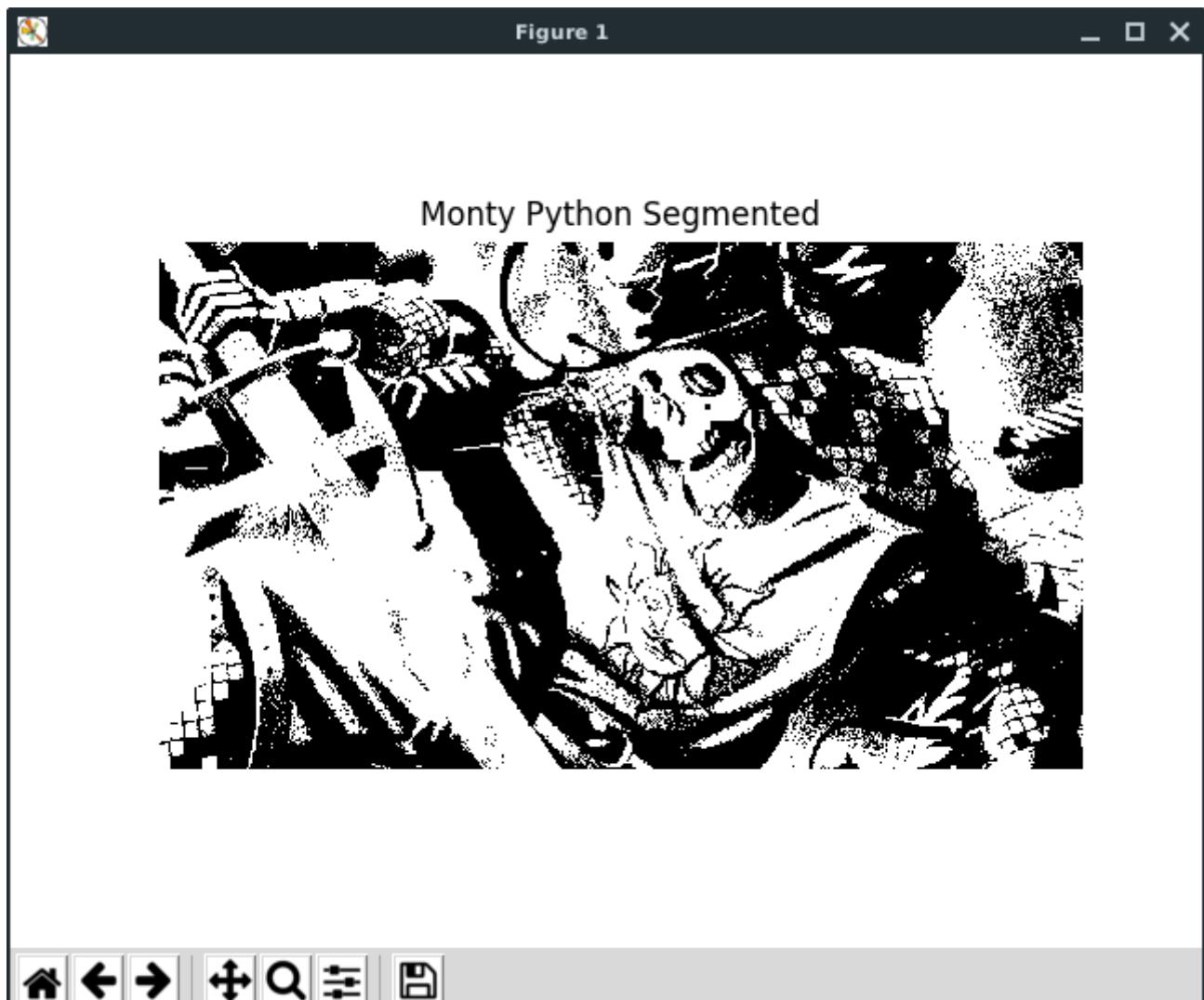
Segmenta a imagem normalizada utilizando a função `threshold_otsu` e seu limiar :

```
1 # Image processing
2
3 from skimage import filters
4
5 # Image segmented and otsu threshold
6
7 otsu_threshold = filters.threshold_otsu(monty_hist)
8
9 monty_segmented = monty_hist > otsu_threshold # Select the objects using the mask
10
11 print(f'Otsu threshold {otsu_threshold}', end='\n\n')
```

OtsuThreshold.py hosted with ❤ by GitHub

[view raw](#)

O resultado da segmentação :



Objetos

Para obter os rótulos dos objetos de uma imagem, utilizaremos a função `label` do módulo Skimage. A função `label` recebe como argumento uma imagem e aplica o conceito de convolução a cada pixel da imagem, a fim de obter os rótulos que distinguem os objetos de uma imagem, realizando a ligação entre os valores de pixels que são iguais entre si. Uma vez que segmentamos a imagem em questão e a mesma se encontra em um formato binário, a função `label` irá separar os objetos do plano de fundo através da diferença entre 0–1, cada objeto possui um rótulo que o distinguir dos demais.

O modo como reconhecemos objetos depende da nossa capacidade de generalização, uma vez que um objeto pode possuir diferentes versões, com pequenas ou grandes diferenças. Cabe a nós generalizar para determinar a qual classe tal objeto pertence. O mesmo princípio pode ser alcançado digitalmente / matematicamente através de convolução. Ao invés de analisarmos um objeto como um todo, analisamos as características específicas, ou seja as partes, que podem indicar a qual classe tal objeto pertence. Em suma, convolução possibilita a extração, summarização e modificação de característica. A convolução ocorre a partir do produto entre uma máscara, arbitrária, ou não, e uma matriz.

Redes neurais profundas convolucionais

Aplique a função `label` utilizando a imagem segmentada na etapa anterior :

```

1 # Image processing
2
3 from skimage import measure
4
5 # Get labels
6
7 monty_label = measure.label(monty_segmented)
8
9 print(monty_label.shape)
```

Label.py hosted with ❤ by GitHub

[view raw](#)

O resultado da extração dos rótulos :

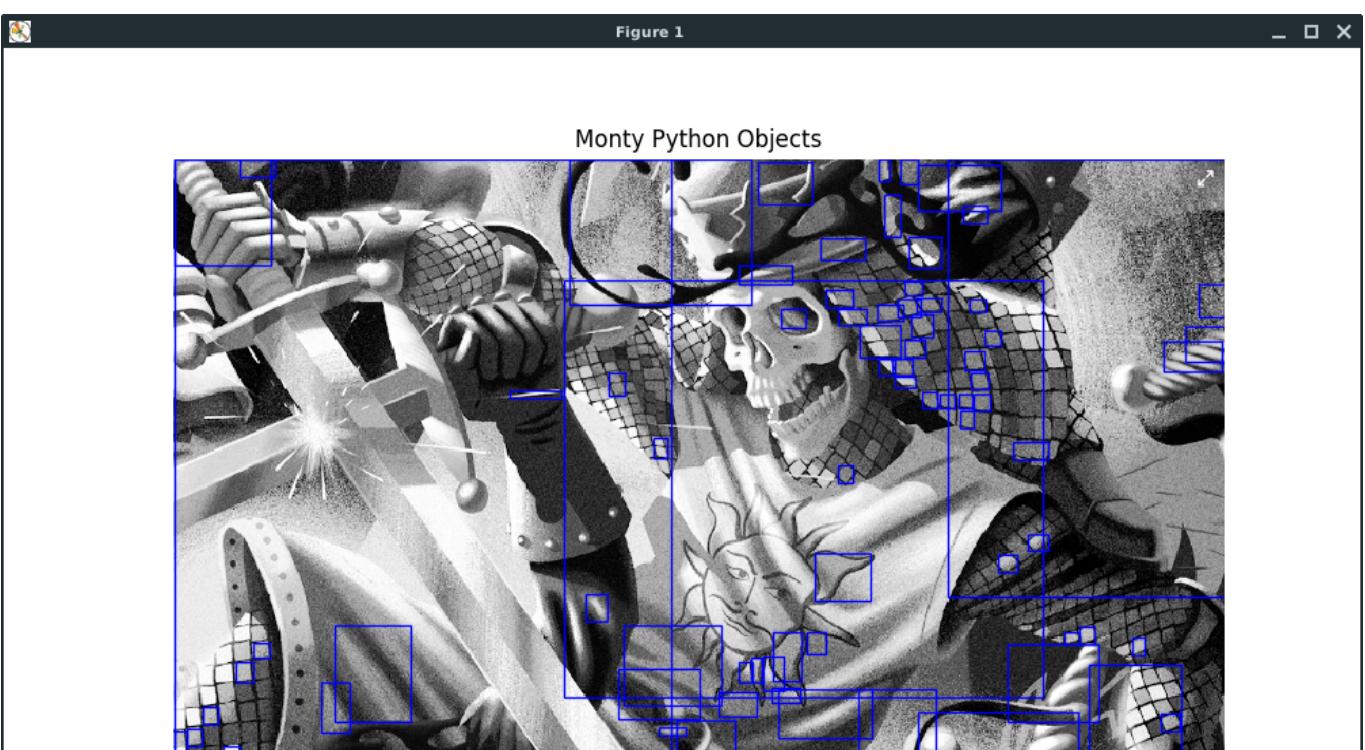
```
1 # Plots
```

```
2
```

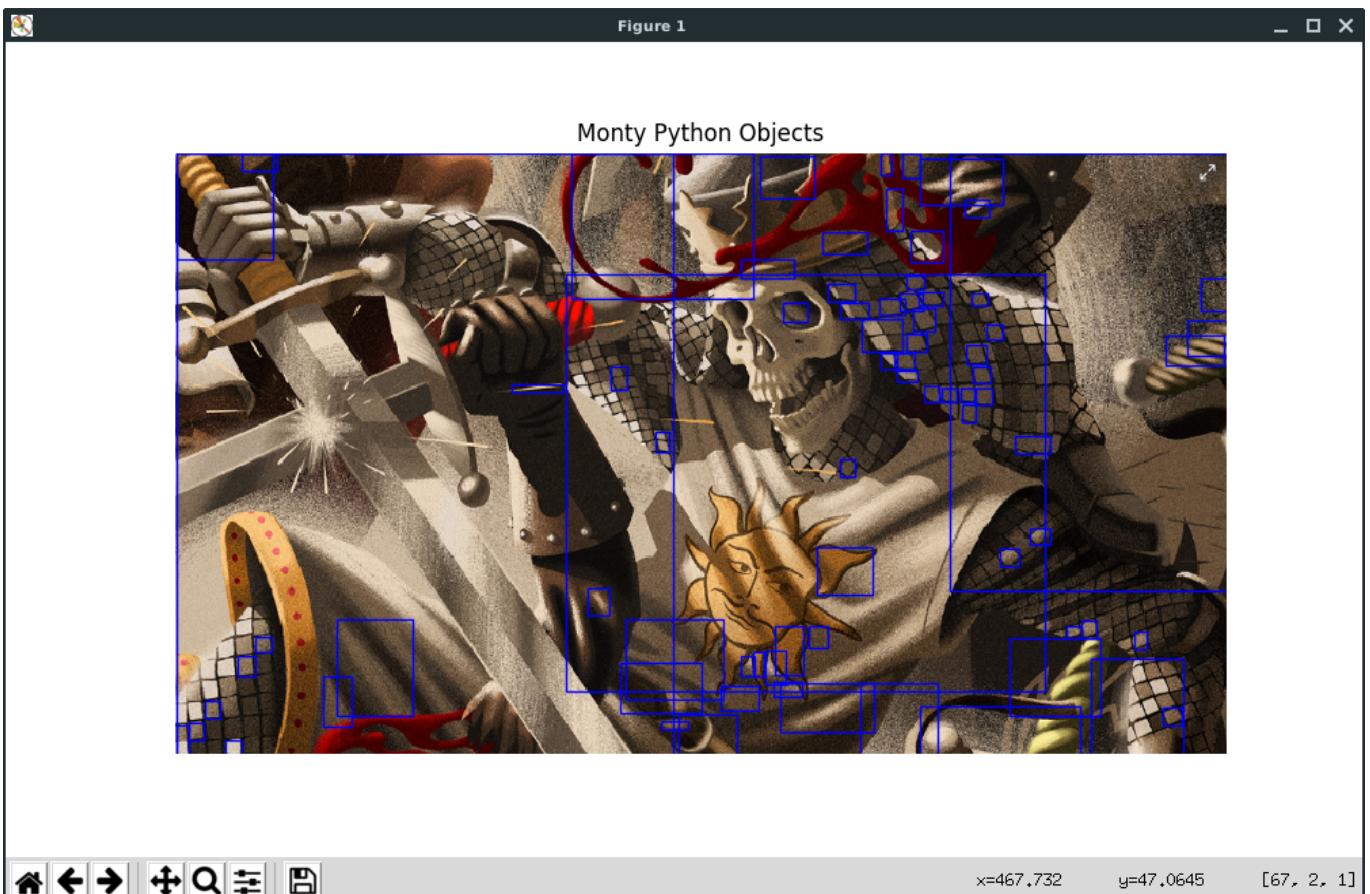
```
~  
3 import matplotlib.patches as patches  
4  
5 # Image processing  
6  
7 from skimage import measure  
8  
9  
10 # Plot objects  
11  
12 fig, ax = pp.subplots(figsize=(10,6))  
13  
14 ax.imshow(monty_hist, cmap='gray')  
15  
16 ax.set_title('Monty Python Objects')  
17  
18 ax.axis('off')  
19  
20 for region in measure.regionprops(monty_label):  
21     # Take large regions  
22  
23     if region.area > 50:  
24         minr, minc, maxr, maxc = region.bbox  
25  
26         rect = patches.Rectangle((minc, minr), maxc - minc, maxr - minr,  
27                                     fill=False, edgecolor='blue')  
28  
29         ax.add_patch(rect)  
30  
31 pp.show()
```

Objects.py hosted with ❤ by GitHub

[view raw](#)



Plot 4 — Monty Python — Rótulos Gray



Plot 5 — Monty Python — Rótulos RGB

Para obter um melhor resultado na segmentação de imagens complexas como a escolhida nesse artigo, é possível aplicar morfologia matemática, como por exemplo através do [Scipy](#), na imagem.

Fim

Isso é tudo pessoal. Se curtiu, por favor, compartilhe e se ficou com dúvida, comente abaixo.

Código

```
1
2 # Image I/O
3
4 import imageio
5
6 # Plots
7
8 import matplotlib.pyplot as pp
```

```
9
10 import matplotlib.patches as patches
11
12 # Image processing
13
14 from skimage import color, img_as_float, exposure, filters, measure
15
16 # Arrangement processing
17
18 import numpy as np
19
20 # Read image
21
22 monty = imageio.imread('monty.png')
23
24 # Convert image to float and gray
25
26 monty_gray = color.rgb2gray(img_as_float(monty))
27
28 # Print shape and data type of images
29
30 print(f'\nRgb shape {monty.shape}, size {len(monty)} and type {monty.dtype}\n')
31
32 print(
33     f'Gray shape {monty_gray.shape}, size {len(monty_gray)} and type {monty_gray.dtype}', end='\n'
34 )
35
36 # Histogram equalization
37
38 monty_hist = exposure.equalize_hist(monty_gray)
39
40 # Image segmented and otsu threshold
41
42 otsu_threshold = filters.threshold_otsu(monty_hist)
43
44 print(f'Otsu threshold {otsu_threshold}', end='\n\n')
45
46 # Select the objects using the mask
47 monty_segmented = monty_hist > otsu_threshold
48
49 # Get labels
50
51 monty_label = measure.label(monty_segmented)
52
53 # Plot objects
54
55 fig, ax = pp.subplots(figsize=(10,6))
56
57 ax.imshow(monty, cmap='gray') # Monty
58
59 ax.set_title('Monty Python Objects')
```

```
59     ax.set_title('Monty Python Objects')
60
61     ax.axis('off')
62
63     for region in measure.regionprops(monty_label):
64         # Take large regions
65
66         if region.area > 50:
67             minr, minc, maxr, maxc = region.bbox
68
69             rect = patches.Rectangle((minc, minr), maxc - minc, maxr - minr,
70                                     fill=False, edgecolor='blue')
71
72             ax.add_patch(rect)
73
74     pp.show()
```

Code.py hosted with ❤ by GitHub

[view raw](#)

Image

Python

Brasil

Programming

Programação

Medium

[About](#) [Help](#) [Legal](#)

Get the Medium app

