

Capítulo

8

Visão computacional em *python* utilizando as bibliotecas *scikit-image* e *scikit-learn*

Romuere R. V. Silva, José G. F. Lopes, Flávio H. D. Araújo, Fátima N. S. Medeiros e Daniela M. Ushizima

Abstract

Computer vision algorithms use matrices as input and produce a more compact information as output. These algorithms play a major role in applications of object classification and scene identification captured digitally. Nowadays, there are several libraries that implement computer vision algorithms. Our focus is to present python-based libraries, such as scikit-image and scikit-learn, that provide functions for digital image processing and machine learning, respectively. These open-source libraries allow writing sophisticate algorithms to solve problems in classification, clusterization, and content-based image retrieval. We illustrate the application of these libraries by describing a computer vision workflow that detects real situations of traffic imprudence from surveillance camera images.

Resumo

Algoritmos de visão computacional utilizam matrizes como entrada e produzem uma informação mais compacta como saída. Esses algoritmos são utilizados em aplicações para classificação de objetos e cenas capturadas digitalmente. Atualmente existem diversas bibliotecas que implementam algoritmos de visão computacional. Nossa foco é apresentar as bibliotecas scikit-image e scikit-learn, que possuem funções de processamento digital de imagens e aprendizado de máquina, respectivamente. Essas bibliotecas de código aberto permitem a escrita de sofisticados algoritmos para resolver problemas em classificação, clusterização e recuperação de imagens baseada em conteúdo. Nós ilustramos a aplicação dessas bibliotecas descrevendo um sistema de visão computacional para um problema real de imprudência no trânsito utilizando imagens de câmeras.

8.1. Introdução

A área de Visão Computacional utiliza modelos de descrição de objetos e cenas capturadas digitalmente para tomada de decisões, automatização de tarefas, seja na indústria de manufatura, farmacêutica, automobilística, dentre outras.

Atualmente existem diversas bibliotecas que implementam algoritmos de visão computacional tais como: *OpenCV* e *Matlab*. No entanto, as opções anteriores apresentam maiores desafios dado que a biblioteca *OpenCV* é geralmente usada por desenvolvedores mais avançados, e a licença do software *Matlab* apresenta custo elevado que inviabiliza seu uso em circunstâncias mais amplas, como em sala de aula, projetos de estudantes e pequenas empresas.

As bibliotecas *scikit-image* e *scikit-learn* são de código aberto e possuem uma extensa documentação com inúmeras funções para viabilizar o processamento digital de imagens, classificação, clusterização e recuperação de imagens baseada em conteúdo. Essas bibliotecas são utilizadas em diversas aplicações que dependem da linguagem *python* para desenvolvimento.

Esse capítulo tem o objetivo de introduzir os principais comandos da linguagem *python* que habilitam o desenvolvedor a realizar processamento de imagens, incluindo a utilização das bibliotecas *scikit-image* e *scikit-learn* aplicadas a um problema real de Visão Computacional.

8.2. Visão Computacional

Com a popularização dos dispositivos eletrônicos para captura de imagens e vídeos, a análise automática desses dados através de programas computacionais tornou-se essencial nas mais variadas áreas de domínio da ciência. Para isso, algoritmos e representações são desenvolvidos para permitir que uma máquina reconheça objetos, pessoas, cenas e atividades. Nestes termos, o computador passa a desempenhar tarefas complexas e ser capaz de reconhecer objetos em figuras e filmes.

De forma geral, algoritmos de visão computacional utilizam matrizes bidimensionais ou hiperdimensionais como entrada e produzem uma informação mais compacta como saída, como por exemplo a classificação de um objeto. No caso de decisões em sistemas de controle de qualidade de materiais, a tomada de decisão geralmente é um módulo no sistema de monitoramento, responsável pela detecção de rachaduras a partir de imagens produzidas durante o processo de deformação para teste de resistência de materiais cerâmicos. A Figura 8.1 ilustra a relação entre as áreas de visão computacional, computação gráfica e processamento digital de imagens. Usualmente estas áreas se confundem, contudo elas se diferenciam principalmente pelo tipo de dados na entrada e saída do sistema.

8.3. Problema Abordado

É extensa a lista de aplicações possíveis dos algoritmos de visão computacional, que são particularmente relevantes na realização de tarefas insalubres e repetitivas que de outra forma seriam delegadas a seres humanos. Outras aplicações são auxiliar na percepção robótica, em agentes autônomos e aprimorar habilidades humanas tais como interface

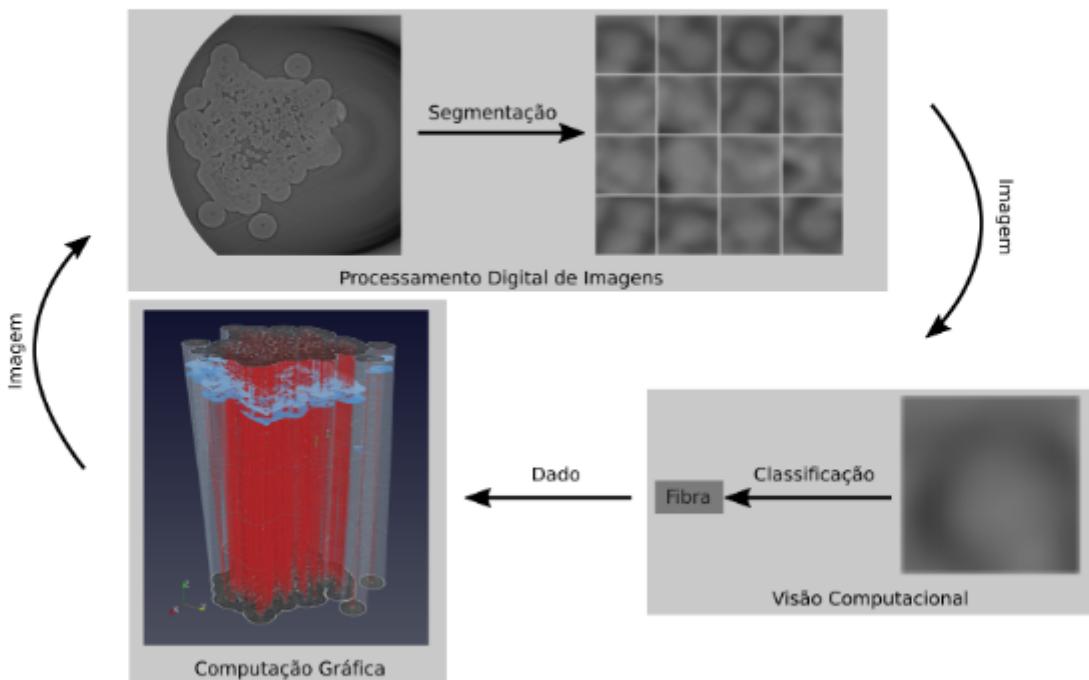


Figura 8.1: Visão Computacional × Computação Gráfica × Processamento Digital de Imagens. A entrada e a saída é o que define cada área. Adaptado de [Ushizima et al. 2014].

humano-computador e visualização.

Dentre as possíveis aplicações em visão computacional utilizaremos imagens de trânsito com foco na detecção de capacete de motociclistas. Alguns fatores determinam o interesse nesse tipo aplicação: 1) o número de acidentes de trânsitos envolvendo motociclistas tem aumentado nos últimos anos em vários países; 2) motocicletas são os veículos que se tornaram mais populares devido ao seu baixo custo; 3) o principal equipamento de segurança é o capacete e, apesar disso, muitos motociclistas não fazem uso do equipamento ou mesmo o fazem incorretamente [Silva et al. 2017]. Utilizaremos uma base de imagens que possui 255 imagens com motociclistas que estão ou não fazendo uso do capacete¹ adequadamente. A partir dessa base será possível realizar experimentos de segmentação para determinação de uma região de interesse, extração e seleção de atributos, e classificação. A Figura 8.2 mostra as principais etapas para a resolução do problema abordado.

8.4. Segmentação de Imagens

O processo de agrupamento dos pixels em uma componente conexa relevante é denominado de segmentação. Esse processo consiste em subdividir uma imagem em regiões ou objetos segundo um critério significativo. Agrupar pixels ou conjuntos de pixels está diretamente relacionado ao problema em estudo e geralmente requer algoritmos iterativos.

A separação dos pixels relativos a cada objeto, ou região, é uma etapa fundamental

¹Imagens disponíveis em <https://github.com/romuere/databases>

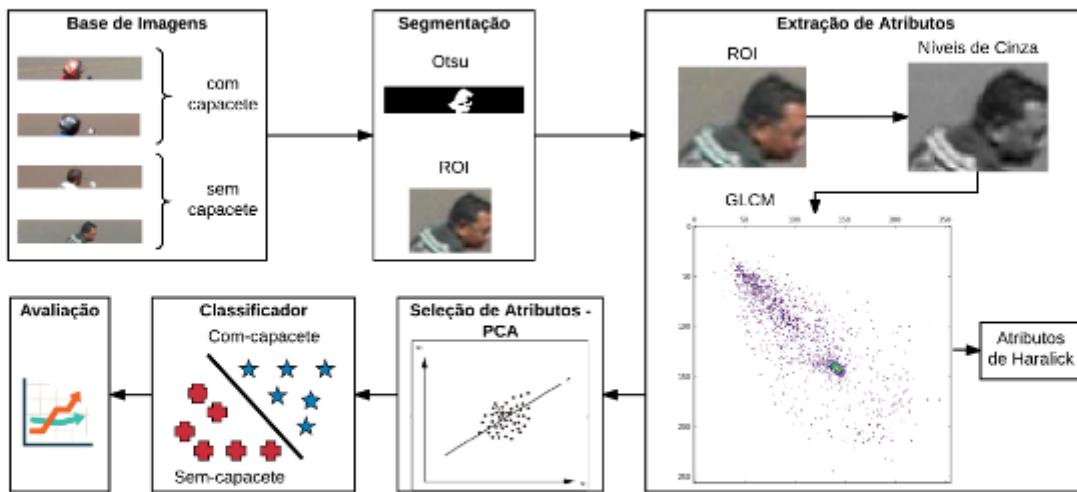


Figura 8.2: Metodologia fundamental para exploração de imagens usando visão computacional conforme proposto nesse tutorial.

para o sucesso da análise da imagem. Embora o ser humano possa facilmente identificar regiões de características semelhantes ou objetos presentes em uma imagem, a realização da mesma tarefa por um computador requer a implementação de algoritmos especialistas em analisar características dos pixels ou da distribuição dos mesmos na imagem.

Em outras palavras, a segmentação é um processo que divide uma região espacial (imagem) R em n sub-regiões. O valor de n varia de acordo com o problema a ser trabalhado. Por exemplo, no processamento e análise de imagens de células, a segmentação frequentemente consiste em identificar três regiões ($n = 3$), a saber, fundo (*background*), citoplasma e núcleo.

Durante o processamento digital de imagens, uma das formas de particionar a imagens em subconjuntos de pixels é realizar limiarização através do método clássico de Otsu [Otsu 1979]. Nesse método, assume-se que a imagem possui duas classes de pixels: de primeiro plano e de fundo. Então, calcula-se o valor ótimo de limiar que separa as classes de modo que a variação intra-classe seja mínima. O cálculo deste limiar é dado por:

$$\sigma_w^2(t) = w_0(t)\sigma_0^2(t) + w_1(t)\sigma_1^2(t), \quad (1)$$

em que os pesos $w_0(t)$ e $w_1(t)$ são as probabilidades das duas classes serem separadas pelo limiar t , sendo as variâncias dessas classes σ_0 e σ_1 .

Para calcular o probabilidade da classe, utiliza-se o histograma com $L = 256$ níveis de cinza da imagem:

$$w_0(t) = \sum_{i=0}^{t-1} p(i) \quad (2)$$

$$w_1(t) = \sum_{i=t}^L p(i)$$

O algoritmo é definido da seguinte forma:

1. Calcule o histograma e as probabilidades para cada nível de intensidade;
2. Selecione os valores iniciais para $w_i(0)$;
3. Passe por todos os valores possíveis de limiar de $t = 0, \dots, 255$;
 - (a) Atualize os valores de w_i ;
 - (b) Calcule $\sigma_j^2(t)$;
4. o valor ótimo será o máximo; $\sigma_j^2(t)$.

Com o valor de limiar calculado, podemos obter uma segmentação como mostrado na Figura 8.3.



Figura 8.3: Exemplar da base de imagens após a segmentação utilizando Otsu.

8.4.1. Código em python

Assim como a maioria das bibliotecas de processamento digital de imagens, o *scikit-image* possui a implementação do limiar de Otsu. O objetivo de utilizar essa rotina foi obter uma Região de Interesse (*Region of Interest* - ROI) com o recorte do segmento contendo a cabeça do motociclista.

O Código Fonte 8.1 mostra os passos para realizar a segmentação de uma imagem baseado no limiar de Otsu. Após transformar a imagem em níveis de cinza em uma imagem binária (Linha 9), é feito um pós-processamento para remover pequenas regiões que foram segmentadas e que não fazem parte da região de interesse (Linhas 10 a 20), como mostra a Figura 8.4 com exemplos dessa etapa. Isso é feito para evitar que pequenas regiões, como cantos ou objetos, que não pertencem à região da cabeça do motociclista, sejam incluídos no recorte. O passo seguinte é fazer o recorte baseado na região segmentada e obter a imagem de onde serão extraídos os atributos para a posterior classificação do segmento (Figura 8.5).

```
1 from skimage.io import imread #função do pacote skimage para leitura de
   #imagens
2 from skimage.color import rgb2grey #função que transforma uma imagem
   #RGB (colorida) em níveis de cinza
3 from skimage.filters import threshold_otsu #função que implementa o
   #limiar de Otsu
4 import numpy as np #pacote python que oferece suporte a arrays e
   #matrizes multidimensionais
5 from skimage.measure import label, regionprops
6
7 path_image = "/home/users/image.png" #endereço da imagem
```

```

9 imagem = imread(path_image) #leitura da imagem
10 grey_image = rgb2grey(imagem)#Converte as imagens em niveis de cinzas
11 otsu = threshold_otsu(grey_image)/255 #Calcula o limiar utilizando o
    metodo de Otsu
12 img_otsu = grey_image < otsu #pixels com valores menores que 'otsu'
    serao brancos , caso contrario serao pretos

13 #----- remove pequenas regioes -----
14 label_img = label(seg, connectivity = grey_image.ndim)#detecta regioes
    nao conectadas
15 props = regionprops(label_img)#calcula propriedade importantes de cada
    regiao encontrada (ex. area)

17 #Convert todas as regioes que possuem um valor de area menor que a
    maior area em background da imagem
18 area = np.asarray([props[i].area for i in range(len(props))])#area de
    cada regiao encontrada
19 max_index = np.argmax(area)#index da maior regiao
20 for i in range(len(props)):
21     if(props[i].area < props[max_index].area):
22         label_img[np.where(label_img == i+1)] = 0#regiao menor que a
            maior eh marcada como background
23
24 label_img = label_img*1#transforma imagem de valores booleanos para
    inteiros
25 #----- recorte da regiao de interesse -----
26 # Obtendo os limites verticais das imagens segmentadas
27 ymin = np.min(np.where(label_img == True)[1])
28 ymax = np.max(np.where(label_img == True)[1])
29 imagem_cortada = imagem[:,ymin:ymax,:]
```

Código Fonte 8.1: Método de Otsu aplicado a duas imagens.



Figura 8.4: Imagem processada após a retirada doe artefatos.

8.5. Extração de Atributos

Atributos de forma, cor e textura são propriedades presentes em objetos registrados na maioria das imagens. Estas características podem ser quantificadas e usadas como descriptores da imagem ou dos objetos numa cena. Esses descriptores compõem um conjunto de propriedades representadas em um vetor de escalares, denominado descriptor da imagem.



Figura 8.5: Exemplares da base de imagens após a determinação da região de interesse.

Assim sendo, cada objeto pode ser representado por um ponto em um espaço R^n , definido em termos das n características extraídas. Em alguns casos, é desejável que os descritores apresentem a propriedade de invariância a transformações afins, ou seja, invariante a rotação, escala, e translação.

Podemos representar uma região de dois modos: através de características externas, pertencentes a fronteira, ou em termos de características internas, pertencentes aos pixels que constituem a região. Escolher o modo de representação consiste em um passo essencial para permitir com que as imagens sejam processadas em um computador. O segundo passo é descrever a região com base na representação escolhida.

Após a extração de atributos, todas as informações obtidas são agrupadas em um vetor de escalares denominado vetor de atributos. A Figura 8.7 mostra um exemplo de extração de atributos.

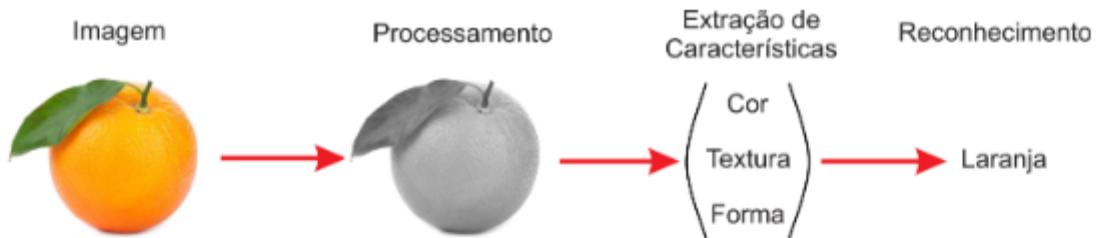


Figura 8.6: Extração de atributos para reconhecimento de padrões.

8.5.1. Matriz de Coocorrência de Níveis de Cinza

A matriz de coocorrência de níveis de cinza, do inglês *Grey-Level Co-occurrence Matrix* (GLCM) é uma técnica que tem como base a análise de textura em imagens. Na GLCM são analisadas as coocorrências existentes entre pares de pixels através de algum padrão. A matriz GLCM é sempre quadrada e armazena as informações das intensidades relativas dos pixels. Por este motivo, as imagens utilizadas são sempre em tons de cinza [Haralick 1973].

As probabilidades de coocorrências são calculadas entre dois níveis de cinza i e j , utilizando uma orientação Θ e uma distância conhecida como espaçamento entre pares de pixels. Essa orientação pode assumir os valores 0° , 45° , 90° e 135° graus. Para cada relacionamento espacial possível (distância e orientação) existe uma matriz de coocorrência. Desse modo, todas as informações sobre a textura de uma imagem estarão contidas nessa matriz [Baraldi and Parmiggiani 1995].

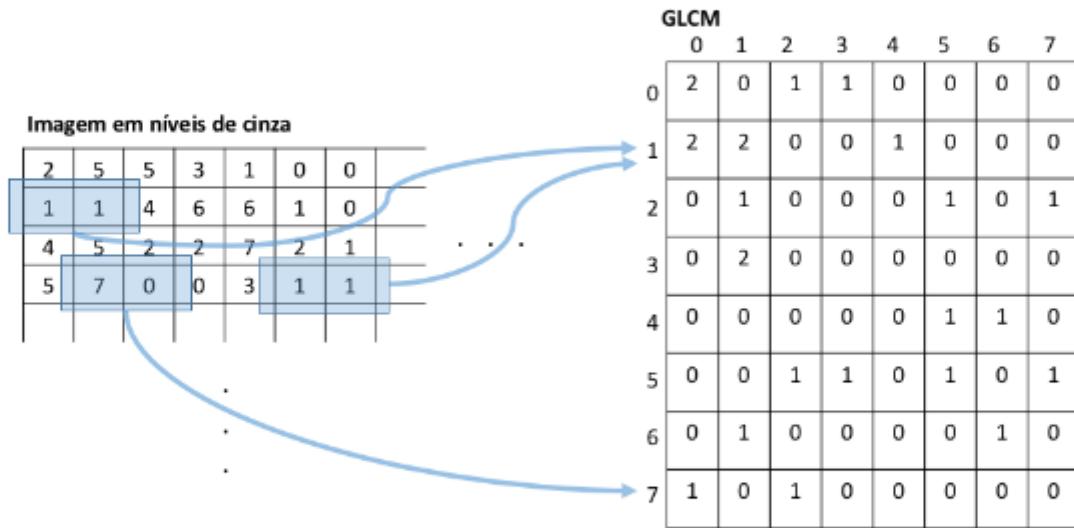


Figura 8.7: Exemplo de cálculo da matriz GLCM com espaçamento entre pares de pixels igual a 1 e $\Theta = 0^\circ$. Intensidade de pixels da imagem (esquerda), GLCM (direita).

A Figura 8.8 mostra a matriz de coocorrência calculada para duas amostras da base de imagens de motociclistas. Pode-se observar a diferença do padrão de espalhamento entre as matrizes (Figuras 8.8(b) e (d)), onde a primeira concentra informação ao longo da diagonal, enquanto a outra apresenta transições entre níveis de cinza mais heterogêneas.

Haralick [Haralick 1973] definiu 14 características significativas para a GLCM há mais de quarenta anos atrás, e muitas outras medidas derivadas dessas foram acrescentadas [Sabino et al. 2004]. Contudo, a utilização de algumas dessas características pode gerar melhor desempenho do que a utilização de todas. Assim, nesse trabalho são feitos cálculos dos seguintes atributos de textura: contraste (Equação 3), dissimilaridade (Equação 4), homogeneidade (Equação 5), energia (Equação 6), correlação (Equação 7) e segundo momento angular (ASM) (Equação 8).

$$\text{Contraste} = \sum_{i,j=0}^{L-1} P_{ij}(i-j)^2, \quad (3)$$

onde L é a quantidade de níveis de cinza e P é a GLCM.

$$\text{Dissimilaridade} = \sum_{i,j=0}^{L-1} P_{ij}|i-j|. \quad (4)$$

$$\text{Homogeneidade} = \sum_{i,j=0}^{L-1} \frac{P_{ij}}{1+(i-j)^2}. \quad (5)$$

$$\text{Energia} = \sqrt{\sum_{i,j=0}^{L-1} (P_{ij})^2}. \quad (6)$$

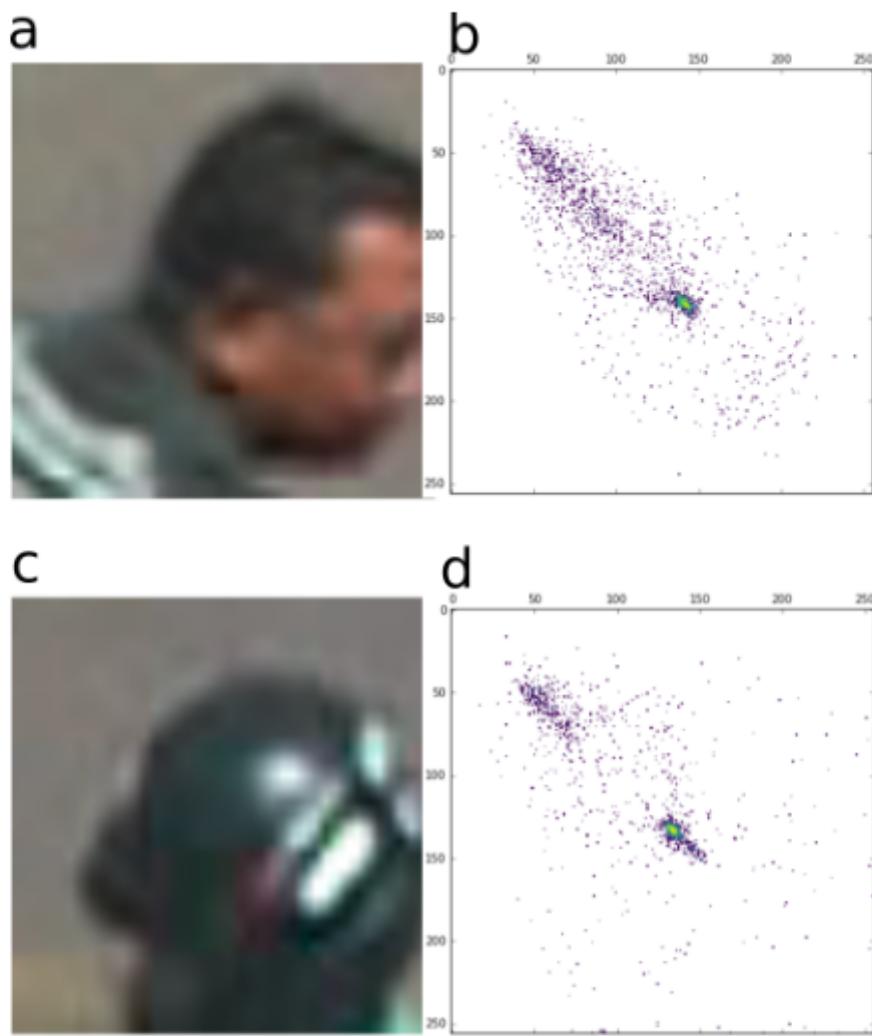


Figura 8.8: Amostras de imagens (a) e (c) e suas respectivas matrizes GLCM (b) e (d).

$$Correlacao = \sum_{i,j=0}^{L-1} \frac{(i - \mu_i)(j - \mu_j)P_{ij}}{\sqrt{(\sigma_i)^2(\sigma_j)^2}}, \quad (7)$$

onde μ é a média e σ é o desvio padrão.

$$ASM = \sum_{i,j=0}^{L-1} (P_{ij})^2. \quad (8)$$

8.5.1.1. Código em *python*

A obtenção da GLCM é realizada em duas etapas sendo a primeira o cálculo da matriz de coocorrências e a segunda o cálculo dos atributos de Haralick a partir dessa matriz. A biblioteca *scikit-image* possui duas funções com esse objetivo: *greycomatrix* é responsável