Arestas e Contornos

Conceito

Detecção de bordas é crucial para interpretação de imagens, pois distingue estruturas e descontinuidades. É extremamente importante no processamento de imagens.

Arestas

Bordas são cruciais na visão e podem reconstruir uma figura. Como surgem e como são localizadas tecnicamente é a questão para border detection.

Bordas são posições onde a intensidade local da imagem muda distintamente em uma orientação particular. Quanto mais forte a mudança de intensidade local, mais evidente é a borda. A primeira derivada de uma função é usada na matemática para medir a quantidade de mudança em relação à distância espacial, e é usada para desenvolver um detector de borda simples.

Detecção de Arestas Baseada no Gradiente

A primeira derivada pode ser usada para detectar bordas em uma imagem. Para aproximar a primeira derivada em uma função discreta (imagem), como um perfil de linha de uma imagem real, um método simples é ajustar uma linha reta aos valores das funções vizinhas. Esse método pode ser aplicado na direção vertical para estimar a primeira derivada ao longo das colunas da imagem

A derivada parcial é a derivada de uma função multidimensional ao longo de um de seus eixos coordenados. O gradiente de uma imagem em um ponto é um vetor formado pelas derivadas parciais da função imagem ao longo dos eixos x e y naquela posição. A magnitude do gradiente é invariável sob a rotação da imagem, permitindo a localização isotrópica de bordas e sendo a base de muitos métodos práticos de detecção de bordas.

A aproximação das primeiras derivadas horizontais pode ser facilmente implementada por um filtro linear com uma matriz de coeficientes como [-1, 0, 1] ou [0.5, 0, -0.5]. O pixel central é ignorado e portanto, o seu novo valor será a variação de tonalidade da imagem na sua posição

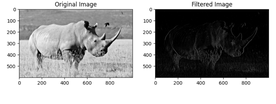
```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

img = cv2.imread('./images/rhino.jpg', cv2.IMREAD_GRAYSCALE)

# create the kernel
kernel = np.array([[-1, 0, 1]])

# apply the gradient filter
filtered_img = cv2.filter2D(img, -1, kernel)


fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.imshow(img, cmap='gray')
ax1.set_title('Original Image')
ax2.imshow(filtered_img, cmap='gray')
ax2.set_title('Filtered Image')
plt.show()
```



Operadores de Sobel e Prewitt

Os operadores de borda Prewitt e Sobel são métodos clássicos similares que utilizam filtros de gradiente para reduzir a sensibilidade dos gradientes a ruídos. Eles diferem apenas nos filtros usados, que têm dimensões de 3x3.

Prewitt usa componentes médios do gradiente

Os filtros para o operador Sobel são quase idênticos; no entanto, a parte de suavização atribui maior peso à linha/coluna central

$$H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } H_y^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -2 & \mathbf{0} & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } H_y^P = \begin{bmatrix} -1 & -2 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Como funciona

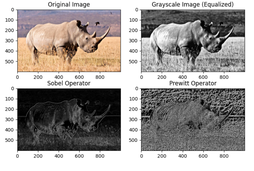
É um método baseado em gradiente que aplica um kernel de convolução à imagem para calcular a magnitude do gradiente em cada pixel. O operador Sobel usa dois kernels 3x3, um para detectar bordas na direção horizontal e outro para detectar bordas na direção vertical.

Para aplicar o operador Sobel a uma imagem, convoluímos cada pixel com ambos os kernels e calculamos a magnitude do gradiente usando a formula de magnitude de gradiente. Se a magnitude do gradiente for maior que um threshold, o pixel é considerado de borda.

```
-1 0 1
-2 0 2
-1 0 1
```

```
-1 -2 -1
 0  0  0
 1  2  1
```

```
G = sqrt(Gx^2 + Gy^2)
```




```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('./images/rhino.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray_eq = cv2.equalizeHist(gray)

# apply the sobel operator
sobelx = cv2.Sobel(gray_eq, cv2.CV_64F, 1, 0, ksize=3)
sobely = cv2.Sobel(gray_eq, cv2.CV_64F, 0, 1, ksize=3)
sobel = np.sqrt(np.square(sobelx) + np.square(sobely))

# apply the prewitt operator
prewittx = cv2.filter2D(gray_eq, -1, np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]))
prewitty = cv2.filter2D(gray_eq, -1, np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]]))
prewitt = np.sqrt(np.square(prewittx) + np.square(prewitty))

# plot
plt.subplot(2, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.subplot(2, 2, 2)
plt.imshow(gray_eq, cmap='gray')
plt.title('Grayscale Image (Equalized)')
plt.subplot(2, 2, 3)
plt.imshow(sobel, cmap='gray')
plt.title('Sobel Operator')
plt.subplot(2, 2, 4)
plt.imshow(prewitt, cmap='gray')
plt.title('Prewitt Operator')
plt.show()
```