Filters - 1

Conceito

Com operações pontuais, novos valores de um pixel dependem apenas desse mesmo pixel, portanto, não é possível alterar nitidez ou suavisar as imagens a partir desses

Exemplo simples: Imagens parecem nítidas em áreas onde há grandes diferenças de intensidade local entre pixels vizinhos, enquanto áreas com diferenças de intensidade suaves parecem borradas. Para suavizar uma imagem, uma ideia inicial é substituir cada pixel pela média de seus oito vizinhos imediatos e seu próprio valor original, calculando assim uma média aritmética dos nove valores.

Uma distinção fundamental entre os filtros e as operações de ponto é que, enquanto as operações de ponto dependem apenas do pixel original na mesma posição para gerar um novo valor de pixel, os filtros usam normalmente mais de um pixel da imagem de origem para calcular cada novo valor de pixel.

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

img = cv2.imread('./images/image.jpeg')

# resize (image used was too big)
new_width = int(img.shape[1] * 0.2)
new_height = int(img.shape[0] * 0.2)
resized_img = cv2.resize(img, (new_width, new_height))

# median blure, 15x15 kernel
blurred_img = cv2.medianBlur(resized_img, 15)

# plot
plt.subplot(121)
plt.imshow(resized_img[:,:,::-1])
plt.title('Resized Image')
plt.xticks([]), plt.yticks([])
plt.subplot(122)
plt.imshow(blurred_img[:,:,::-1])
plt.title('Blurred Image')
plt.xticks([]), plt.yticks([])
plt.show()
```

Box FIlter

O box filter é o filtro mais simples de suavização, mas não é considerado um bom filtro devido ao seu comportamento no domínio da frequência. Ele não é isotrópico, não considera a distancia dos pixels visinhos ao calcular o blur

Filtro de Suavização Gausiana

O filtro gaussiano é uma alternativa superior, correspondendo a uma função gaussiana bidimensional que coloca mais ênfase no pixel central do que nos pixels circundantes. O filtro gaussiano é isotrópico e apropriado para ser aplicado no domínio da frequência.

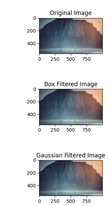
Ambos os filtros requerem loops aninhados para iterar sobre as coordenadas da imagem e da região do filtro.

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


img = cv2.imread('./images/image2.jpeg')

# box filter
box_filtered = cv2.boxFilter(img, -1, (13, 13))
# gausian filter
gaussian_filtered = cv2.GaussianBlur(img, (13, 13), 0)

# plot
fig, ax = plt.subplots(3, 1, figsize=(15, 5))
ax[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
ax[0].set_title('Original Image')
ax[1].imshow(cv2.cvtColor(box_filtered, cv2.COLOR_BGR2RGB))
ax[1].set_title('Box Filtered Image')
ax[2].imshow(cv2.cvtColor(gaussian_filtered, cv2.COLOR_BGR2RGB))
ax[2].set_title('Gaussian Filtered Image')
fig.subplots_adjust(hspace=1.0)
plt.show()
```



Ruído

O ruído é um sinal inesperado nas imagens que distorce a imagem original introduzindo variações aleatórias de brilho, iluminação ou informações de cor.

Existem três tipos comuns de ruído em imagens: Sal e pimenta, impulso e ruído gaussiano. O ruído de sal e pimenta é caracterizado por pixels pretos (pimenta) e brancos (sal) aleatórios na imagem. O ruído de impulso apresenta apenas pixels brancos aleatórios. O ruído gaussiano é o tipo mais comum e apresenta variações na intensidade de pixel seguindo uma distribuição gaussiana.

- Aumento de dados: Adicionar ruído a uma imagem pode ser usado como técnica de aumento de dados para aumentar o tamanho de um conjunto de dados de treinamento para um modelo de aprendizado de máquina. Isso ajuda a reduzir o overfitting e melhorar o desempenho de generalização do modelo.
- Simulação: Em alguns casos, o ruído é adicionado para simular condições do mundo real. Por exemplo, ao testar o desempenho de algoritmos de processamento de imagem, pode ser útil adicionar ruído às imagens para simular as condições nas quais o algoritmo será usado.
- Privacidade: Adicionar ruído a uma imagem pode ser usado como técnica de preservação de privacidade para proteger informações sensíveis na imagem. Por exemplo, adicionar ruído a uma imagem de rosto pode dificultar a identificação da pessoa na imagem por algoritmos de reconhecimento facial.
- Remoção de ruído: Em alguns casos, adicionar ruído a uma imagem pode ser usado para melhorar a qualidade da imagem. Isso pode parecer contraditório, mas em alguns casos, adicionar ruído pode ajudar a remover outros ruídos da imagem. Esse é o fundamento de muitas técnicas de remoção de ruído, como o filtro não local.
- Estético: Adicionar ruído a uma imagem pode ser feito por motivos estéticos. Isso pode dar à imagem uma sensação vintage ou de filme e pode ser usado para criar efeitos visuais interessantes.

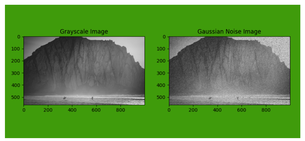
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt


img = cv2.imread('./images/image2.jpeg')

# conver to gray scale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# gausian noise
gaussian_noise = np.zeros_like(gray)
cv2.randn(gaussian_noise, 0, 50)
noisy_image = gray + gaussian_noise

# plot
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
fig.patch.set_facecolor('xkcd:grass green')
ax[0].imshow(gray, cmap='gray')
ax[0].set_title('Grayscale Image')
ax[1].imshow(noisy_image, cmap='gray')
ax[1].set_title('Gaussian Noise Image')
plt.show()
```



Convolução Linear

A convolução linear é a operação matemática usada nos filtros

A convolução linear é uma operação matemática fundamental em processamento de sinais e processamento de imagens. É usada para processar sinais em um domínio de tempo contínuo ou discreto, e é uma operação que envolve a combinação de dois sinais para produzir um terceiro sinal. A operação de convolução linear pode ser definida como a integral ou soma ponderada dos produtos de cada valor do sinal de entrada com os coeficientes do filtro, deslocados ao longo do sinal. Em outras palavras, a convolução linear envolve o deslocamento do filtro ao longo do sinal de entrada e a multiplicação do filtro pelos valores do sinal de entrada correspondentes em cada posição.

Propriedades da Convolução Linear

A comutatividade é uma propriedade matemática importante da convolução linear. Ela estabelece que o resultado é o mesmo, independentemente da ordem em que a imagem e o kernel do filtro são convolvidos. Em outras palavras, não importa se convolvemos a imagem com o kernel ou o kernel com a imagem - o resultado final será o mesmo. Essa propriedade é muito útil em diversas aplicações da convolução linear.

Linearidade: Se multiplicarmos uma imagem por uma constante escalar, o resultado da convolução também será multiplicado pelo mesmo fator. Da mesma forma, se adicionarmos duas imagens pixel a pixel e convolvermos a imagem resultante com um kernel, o mesmo resultado pode ser obtido convolvendo cada imagem individualmente e adicionando os resultados posteriormente. No entanto, adicionar um valor constante à imagem não segue a propriedade de linearidade. Embora a linearidade seja uma propriedade teórica importante, na prática os filtros lineares podem não ser totalmente lineares devido a erros de arredondamento ou limitações de valores de saída.

A associatividade é outra propriedade matemática importante da convolução linear. Ela estabelece que a ordem das operações de filtro sucessivas é irrelevante, ou seja, vários filtros sucessivos podem ser aplicados em qualquer ordem e vários filtros podem ser combinados arbitrariamente em novos filtros. Isso permite a criação de cadeias complexas de filtros para obter resultados específicos e personalizados.

A propriedade de separabilidade é particularmente relevante no processamento de imagens, pois permite que um kernel bidimensional seja separado em dois kernels unidimensionais, o que resulta em economias computacionais significativas. Os filtros gaussianos separáveis são um exemplo de filtros separáveis que são computacionalmente eficientes e rotacionalmente simétricos. Eles são amplamente usados para filtragem de ruído gaussiano e down-sampling de imagens.

Filtros Lineares

Filtros lineares combinam valores de pixel de forma linear, usando um somatório ponderado na região de suporte. O processo de média local é um exemplo onde todos os pixels são adicionados com pesos idênticos. Diferentes filtros podem ser definidos modificando a distribuição dos pesos individuais.

Kernel (Matriz de Filtragem)

A matriz ou máscara do filtro especifica o tamanho, forma e pesos dos pixels da região de suporte para qualquer filtro linear. A matriz tem o mesmo tamanho que a região do filtro e cada elemento representa o peso do pixel correspondente na soma. A função do filtro é uma função de valor real discreta bidimensional e tem um sistema de coordenadas próprio com uma origem, geralmente localizada no centro. O filtro é considerado zero fora da região definida pela matriz.

O tamanho do filtro é importante porque determina a extensão espacial do filtro, ou seja, quantos pixels originais contribuem para cada valor de pixel resultante. Um filtro com suporte maior, como 5x5 ou 21x21 pixels, terá um efeito de suavização mais forte do que um filtro com suporte menor, como 3x3.

A forma da região do filtro não precisa ser quadrada ou retangular e pode ser preferível uma forma circular para um efeito de desfoque isotrópico. Pode-se atribuir pesos diferentes aos pixels na região de suporte para dar maior ênfase aos pixels mais próximos ao centro. A região de suporte pode ser em forma de anel e não precisa conter o próprio pixel original. É necessário um método mais sistemático para especificar e aplicar filtros de maneira direcionada.

Filtros Diferenciais

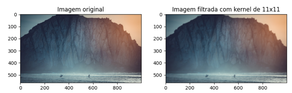
Filtros diferenciais podem ter coeficientes negativos e são usados para calcular a diferença entre somas ponderadas de pixels com coeficientes positivos e negativos na região do filtro. Eles são usados para detecção de bordas e aumento de nitidez de imagem.

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


img = cv2.imread('./images/image2.jpeg')

# diferential 11x11 filter
kernel_size = 11
kernel = cv2.getGaussianKernel(kernel_size, 0)
kernel = kernel * kernel.T
filtered_img = cv2.filter2D(img, -1, kernel)

# plot
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
ax[0].set_title('Imagem original')
ax[1].imshow(cv2.cvtColor(filtered_img, cv2.COLOR_BGR2RGB))
ax[1].set_title('Imagem filtrada com kernel de 11x11')
plt.show()
```




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

img = cv2.imread('./images/image.jpeg')

# resize (image used was too big)
new_width = int(img.shape[1] * 0.2)
new_height = int(img.shape[0] * 0.2)
resized_img = cv2.resize(img, (new_width, new_height))

# create list of kernel sizes
kernel_sizes = [5, 15, 21, 25]

# create a 2x2 subplot figure with blue background
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10,10), facecolor='skyblue')

# iterate through kernel sizes and apply median blur to each
for i, size in enumerate(kernel_sizes):
    row = i // 2
    col = i % 2
    kernel = np.ones((size, size), np.float32) / (size * size)
    blurred_img = cv2.filter2D(resized_img, -1, kernel)
    axs[row][col].imshow(blurred_img[:,:,::-1])
    axs[row][col].set_title(f'{size}x{size} Kernel')
    axs[row][col].set_xticks([]), axs[row][col].set_yticks([])

# show the plot
plt.show()
```