CCM-206 - INTELIGÊNCIA NA WEB E BIG DATA

Bayard da Rocha – RA: 21201810004 – 2018.1

PROVA – Questão Prática: *"Aplicação do Filtro de Convolução em Imagens Gigantescas"*

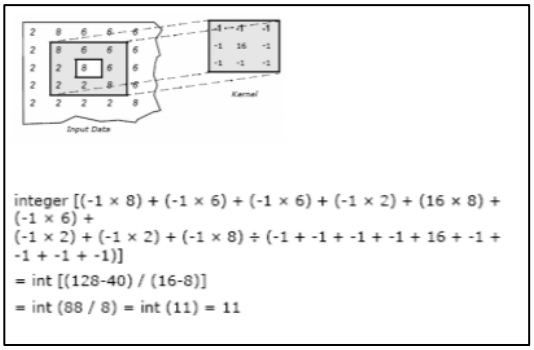
Optei pela implementação do algoritmo para aplicarmos Filtro de Convolução aplicada em imagens. Convolução é um efeito de filtro de propósito geral para imagens (desfocagem, nitidez, mapa de relevo, dentre outros).

A técnica consiste em aplicar uma matriz que chamamos de Kernel a uma matriz geradora da Imagem através de operação matemática composto por inteiros; funciona determinando o valor de um pixel central adicionando valores ponderados de todos os seus vizinhos juntos onde a saída é uma nova imagem filtrada modificada.

A formula abaixo corresponde à formula de Convolução para a obtenção do novo valor do Pixel alvo:

$$V = \left| \frac{\sum_{i=1}^{\cdot} \left( \sum_{j=1}^{\cdot} f_{ij} d_{ij} \right)}{F} \right|$$

Para o Pixel alvo, realizamos a multiplicação dos pixels vizinhos pelo valor na Kernel na mesma posição ; realiza-se a somatória de todas as multiplicações e o resultado divide-se pela somatória dos valores que compões a Kernel. Graficamente podemos representar conforma abaixo.




Busquei referencias de Imagens Gigantes para poder fazer um comparativo entre o tempo de processamento para a aplicação do Filtro de Convolução utilizando algoritmo sequencial e o mesmo filtro utilizando técnica de paralelismo.

A seguir segue um roteiro do que foi feito:

**Processamento sequencial**

1) Seleção da Imagem gigante de alta resolução . utilizarei uma de dimensão 2041X1080 pixels
2) Esta imagem gera 3 matrizes (r,g,b) com dimensões (2041x1080)
3) Montei Kernel de 3x3

Abaixo o print do processo:

```
estrutura da foto) =  (1080, 2401, 3)
dimensão b =  (1080, 2401)
dimensão g =  (1080, 2401)
dimensão r =  (1080, 2401)
kernel
[[0 1 0]
 [1 1 1]
 [0 1 0]]
```

Segue algoritmo aplicado que representa a formula acima descrita

```
qtdeOperac = 0
for a in range (np.array(img_foto).shape [2]):
    matrizConvolu = np.copy(matrizBGR[a])
    for x in range (matrizBGR[a].shape [0]-2):
            for y in range (matrizBGR[a].shape [1]-2):
                        subMatriz = matrizBGR[a][slice(x,x+3),slice (y,y+3)] # cria uma subMatriz com as lir
                        matrizConvolu [x+1][y+1] = int(sum(np.reshape(subMatriz*kernel,-1)) / somaKernel)
                        if  matrizConvolu [x+1][y+1] < 0:
                                print (matrizConvolu [x+1][y+1])
                                matrizConvolu [x+1][y+1] = 0
                        qtdeOperac = qtdeOperac + 1
    matrizBGR[a] = matrizConvolu
```

Sua execução resultou na execução de 7.758.366 cálculos de Pixels Alvo ; o tempo total de execução foi de 65,833 segs.

**Processamento paralelo**

O próximo passo agora, contempla a implementação do paralelismo, com a geração de submatrizes 3X3 onde serão aplicados a Kernel dentro do RDD.

O RDD deverá ter a seguinte estrutura :

(coord, rgb) -> (coord_transformada, coord_submatriz, rgb)

Tenho a expectativa de significativa redução no tempo de processamento

Imagem
https://www.google.com.br/search?rlz=1C1EKKP_enBR792BR792&biw=1689&bih=731&tbm=isch&sa=1&ei=P8HrWrXgBYKZwASY_oXYCQ&q=imagens+gigantes+de+alta+resolu%C3%A7%C3%A3o+los+angeles&oq=imagens+gigantes+de+alta+resolu%C3%A7%C3%A3o+los+angeles&gs_l=psy-ab.3...7155.9731.0.10292.12.12.0.0.0.0.126.1001.0j9.9.0....0...1c.1.64.psy-ab..3.0.0....0.rpu5eXfTFMM#imgrc=QqgdEWQ0g2C57M:

Documentos entregue na prova
https://github.com/BayarddaRocha/BIGDATA2018