scc0251/5830— Prof. Moacir Ponti

PAE/monitor: Fernando Pereira dos Santos, Eduardo Santos Carlos Souza

Trabalho 4: filtragem 2D

Desenvolva seu código sem olhar o de colegas. Plágio não será tolerado.

Nesse trabalho você deverá implementar três filtragens 2D distintas, considerando a imagem como uma matriz bidimensional, a transformada discreta de Fourier 2D e o algoritmo 1-NN.

Você deverá implementar usando `python 3` com as bibliotecas `numpy` e `imageio`.

Leia com calma as instruções de cada etapa.

Seu programa deverá permitir receber o nome de uma imagem, a partir da qual você irá aplicar um dos três métodos de filtragem 2D escolhidos (arbitrário, laplaciano da gaussiana, ou sobel), seguido por cortes na matriz resultante para extrair características da imagem. Depois você deverá utilizar essas características para classificar a imagem utilizando o algoritmo 1-NN.

Entrada

1. **Receber via teclado**:
   a) nome da imagem para filtragem (I);
   b) opção de escolha do método (1 – arbitrário, 2 – laplaciana da gaussiana, 3 – operador sobel);
   c) parâmetro(s) que define(m) os pesos do filtro:
      – para o método 1:
      1.a) altura e largura do filtro (h e w)
      1.b) h linhas com w números reais cada, que são os pesos do filtro
      – para o método 2:
      2.a) tamanho do filtro (n)
      2.b) desvio padrão da distribuição (σ)
      – para o método 3: Não há parâmetros
   d) 4 números reais que darão as posições para realização de cortes (*Hlb*, *Hub*, *Wlb* e *Wub*)
   e) nome do arquivo .npy com o *dataset*
   f) nome do arquivo .npy com as *labels* do *dataset*

Método de extração de características

Opção 1: Filtro Arbitrário

Você deverá implementar uma convolução utilizando a matriz arbitrária recebida por teclado. A convolução deve ser realizada no domínio da frequência. Portanto, como no trabalho anterior, o método é baseado no teorema da convolução:

$$w(x) * f(x) \equiv \mathfrak{F}^{-1}(W(u) \cdot F(u)),$$

Então, você deverá realizar a transformada de Fourier bidimensional na imagem e no filtro e, em seguida, realizar o produto ponto-a-ponto das transformadas. No entanto, você **não deverá realizar a transformada inversa**. O resultado desse processo é *Iout*.

Observação 1: Lembre-se que no domínio de frequências o filtro deve ter o mesmo tamanho da imagem, para isso preencha as demais coordenadas com zero.

Opção 2: Filtro Laplaciana da Gaussiana

O processo para esse filtro será idêntico a opção 1 (domínio da frequência). No entanto, ao invés de utilizar um filtro dado por entrada, será utilizado como filtro a matriz M dada pela seguinte equação:

$$LoG_{2D}(x, y, \sigma) = -\frac{1}{\pi\sigma^4} \cdot \left(1 - \frac{x^2 + y^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

Para gerar a matriz você deverá considerar que ela representa a função LoG na região de um quadrado de lado 10 ao redor da origem(0, 0). Ou seja, o centro da matriz está em cima da origem (0, 0), e os extremos da matriz estão em (5, 5), (5, -5), (-5, 5), (-5, -5). Os pontos da matriz entre os extremos devem estar mapeados linearmente entre -5 e 5.

Ex. se um filtro tiver tamanho n=3, então:

$$M = \begin{bmatrix} LoG(-5, 5, \sigma) & LoG(0, 5, \sigma) & LoG(5, 5, \sigma) \\ LoG(-5, 0, \sigma) & LoG(0, 0, \sigma) & LoG(5, 0, \sigma) \\ LoG(-5, -5, \sigma) & LoG(0, -5, \sigma) & LoG(5, -5, \sigma) \end{bmatrix},$$

Ex. se um filtro tiver tamanho n=4, então:

$$M = \begin{bmatrix} LoG(-5, 5, \sigma) & LoG(-1.66, 5, \sigma) & LoG(1.66, 5, \sigma) & LoG(5, 5, \sigma) \\ LoG(-5, 1.66, \sigma) & LoG(-1.66, 1.66, \sigma) & LoG(1.66, 1.66, \sigma) & LoG(5, 1.66, \sigma) \\ LoG(-5, -1.66, \sigma) & LoG(-1.66, -1.66, \sigma) & LoG(1.66, -1.66, \sigma) & LoG(5, -1.66, \sigma) \\ LoG(-5, -5, \sigma) & LoG(-1.66, -5, \sigma) & LoG(1.66, -5, \sigma) & LoG(5, -5, \sigma) \end{bmatrix},$$

Ex. se um filtro tiver tamanho n=5, então:

$$M = \begin{bmatrix} LoG(-5,5,\sigma) & LoG(-2.5,5,\sigma) & LoG(0,5,\sigma) & LoG(2.5,5,\sigma) & LoG(5,5,\sigma) \\ LoG(-5,2.5,\sigma) & LoG(-2.5,2.5,\sigma) & LoG(0,2.5,\sigma) & LoG(2.5,2.5,\sigma) & LoG(5,2.5,\sigma) \\ LoG(-5,0,\sigma) & LoG(-2.5,0,\sigma) & LoG(0,0,\sigma) & LoG(2.5,0,\sigma) & LoG(5,0,\sigma) \\ LoG(-5,-2.5,\sigma) & LoG(-2.5,-2.5,\sigma) & LoG(0,-2.5,\sigma) & LoG(2.5,-2.5,\sigma) & LoG(5,-2.5,\sigma) \\ LoG(-5,-5,\sigma) & LoG(-2.5,-5,\sigma) & LoG(0,-5,\sigma) & LoG(2.5,-5,\sigma) & LoG(5,-5,\sigma) \end{bmatrix},$$

Além disso, você deve normalizar a matriz de forma a garantir que os pesos tenham soma zero, i.e., $\sum \mathbf{w}_{LoG} = 0$. Para isso, realize a soma separadamente dos valores positivos e negativos do filtro gerado e, logo após, a normalização é obtida conforme:

$$filtroNormalizado[x,y] = filtro[x,y] * (-positivos/negativos), onde filtro[x,y] < 0$$

$$filtroNormalizado[x,y] = filtro[x,y], c.c.$$

O resultado da imagem após a convolução é *Iout*.

Opção 3: Operador Sobel

Esse método é consideravelmente diferente dos 2 anteriores. Nele você irá primeiro realizar 2 convoluções na imagem de entrada, gerando 2 imagens diferentes, *Ix* e *Iy*. Essas convoluções não devem ser feitas no domínio da frequência, mas sim no domínio do espaço. As imagens serão gerada utilizando os seguintes filtros:

$$Fx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$Fy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix},$$

Portanto $Ix = I * Fx$ e $Iy = I * Fy$. Em seguida será gerada a imagem de saída *Iout*, a partir da seguinte equação.

$$Iout(i,j) = \sqrt{Ix(i,j)^2 + Iy(i,j)^2}$$

Iout é a saída do operador Sobel. Para esse filtro, no entanto, você deverá aplicar a transformada de Fourier 2D em *Iout*.

Observação 1: Para realizar o produto ponto-a-ponto, preencha o restante dos valores do filtro com zero antes de realizar a transformada.

Observação 2: Nesse trabalho **será permitido** o uso das transformadas de Fourier da biblioteca numpy.

Observação 3: Para realizar a convolução nas bordas no caso do operador Sobel, considere que os pontos fora da imagem contem 0(*zero-padding.*)

Cortes

No final de qualquer um dos três métodos obtém-se uma matriz resultante no domínio da frequência. Agora você deverá realizar cortes nessa matriz para melhorar e simplificar a classificação. Primeiramente, por ser simétrica, você apenas precisa de 1/4 da matriz. Sendo H a altura de Iout e W a largura, obtém-se *Icut1* com:

$$Icut1 = Iout\left[0 : \frac{H}{2}, 0 : \frac{W}{2}\right]$$

Agora, é necessário fazer os cortes dados por *Hlb*, *Hub*, *Wlb* e *Wub* na entrada. *Hlb* define o limite vertical inferior, *Hub* define o limite vertical superior, *Wlb* define o limite horizontal inferior e *Wub* define o limite horizontal superior. Esses são números reais entre 0.0 e 1.0 que ditam os limitantes para o corte (através da multiplicação pelo tamanho de *Icut1*). Então, por exemplo, se *Icut1* é uma matriz 20x20 e *Hlb*, *Hub*, *Wlb*, *Wub* são respectivamente 0.1, 0.2, 0.3 e 0.4, *Icut2* é dada por:

$$Icut2 = Icut1\,[2 : 4, 6 : 8]$$

Classificação

A classificação para esse trabalho utilizará o algoritmo 1-NN. Para facilitar a implementação, K sempre será igual a 1. Primeiro é necessário transformar a matriz resultante dos cortes (*Icut2*) em um vetor (*Vout*), utilizando o mesmo método que o trabalho anterior. Sendo *dim* o tamanho do vetor *Vout* e *ex* o número de exemplos do *dataset*, as dimensões do arquivo .npy de entrada serão [ex, dim]. A classificação será simplesmente a *label* do ponto P (um dos *ex* exemplos do dataset) mais próximo (por distância Euclidiana) do vetor obtido.

Por exemplo, suponha que a imagem resultante dos cortes (*Icut2*) tenha tamanho 5 X 3, ou seja, o vetor (*Vout*) será de tamanho 15. Desta forma, o *dataset* de entrada possuirá n linhas de 15 colunas cada. Com isso, é possível descobrir qual amostra do *dataset* é mais similar do vetor resultante dos cortes aplicando a distância Euclidiana entre dois vetores. O índice desta amostra (número da linha) será o índice que identificará o conteúdo contido no *label* de classificação.

Observação 1: Utilize numpy.load() para carregar o *dataset*.

Saída

A saída será a *label* e o índice de P.

Exemplos de entrada e saída

Exemplo de entrada 1: imagem de entrada, filtro arbitrário, tamanho $h = 3, w = 3$, pesos (número de pesos é igual ao tamanho), cortes 0.1, 0.2, 0.3 e 0.4, dataset mnist.npy e labels mnist_labels.npy.

```
cat.png
1
3 3
7.4 5.2 2.1
0.2 0.2 0.1
0.3 0.9 6.1
0.1 0.2 0.3 0.4
mnist.npy
mnist_labels.npy
```

Exemplo de entrada 2: imagem de entrada, filtro laplaciano da gaussiana, tamanho $n = 5$, $\sigma = 1.7$, cortes 0.1, 0.2, 0.3 e 0.4, dataset mnist.npy e labels mnist_labels.npy.

```
cat.png
2
5
1.7
0.1 0.2 0.3 0.4
mnist.npy
mnist_labels.npy
```

Exemplo de entrada 3: imagem de entrada, operador sobel, cortes 0.1, 0.2, 0.3 e 0.4, dataset mnist.npy e labels mnist_labels.npy.

```
cat.png
3
0.1 0.2 0.3 0.4
mnist.npy
mnist_labels.npy
```

Exemplos de saída: Label e índice de P

```
3
3452
```

Submissão e instruções

No sistema Run.Codes deve incluir apenas o arquivo `.py`

1. **É obrigatório comentar seu código**. Como cabeçalho insira seu nome, número USP, código da disciplina, ano/semestre e o título do trabalho. Haverá desconto na nota caso esse cabeçalho esteja faltando, além de descontos para falta de comentários.

2. **É obrigatório organizar seu código em funções**.