Introdução ao Processamento de Imagens
Trabalho 1
20/08/2025

Gabriel Gomes Marchesan - 202111722011

1 Introdução.

Neste trabalho você explorará as técnicas de processamento de imagens discutidas em aula. As tarefas incluirão perguntas conceituais, atividades de programação e discussão de resultados. O trabalho deve ser feito individualmente.

2 Operações Básicas com Imagens.

2.1 Usando a biblioteca Opencv **cv2.imread()**, implemente a função `read_img(path, grayscale=True)` no arquivo `common.py`. A função deve carregar uma imagem a partir do caminho (`path`) especificado e, opcionalmente, convertê-la para escala de cinza. A imagem carregada deve ser retornada como um array NumPy com valores do tipo float normalizado para o intervalo [0, 1].

Respondida no notebook

2.2 Usando a função `cv2.imwrite()` da biblioteca Opencv, implemente no arquivo `common.py` a função `save_img(img, path)`. Ela deve rescalar o conteúdo da imagem para valores entre 0 e 255 e então salva-la no caminho (**path**) especificado.

Respondida no notebook

2.3 Usando a função `read_img()`, leia a imagem **lenagray.jpg**. Defina uma nova imagem `J`, 256 × 256 da seguinte forma: a metade esquerda de `J`, ou seja, as primeiras 128 colunas, devem ser iguais à metade direita da imagem **lenagray.jpg**. A metade direita de `J`, ou seja, as colunas de 128 colunas restantes, devem ser iguais à metade esquerda da imagem do **lenagray.jpg**. Salve a imagem modificada usando `save_img()`.

Respondida no notebook

2.4 Usando a função `read_img()`, leia a imagem **lenacolor.jpg**. Defina uma nova imagem `J`, com as mesmas dimensões da imagem original. Faça com que o canal vermelho da nova imagem seja igual ao canal azul da imagem original, faça o canal verde da nova imagem igual ao canal vermelho da imagem original, e faça o canal azul da nova imagem igual ao canal verde da imagem original. Salve a nova imagem usando `save_img()`.

Respondida no notebook

3 Transformações de Intensidade.

3.1 Matematicamente, que a transformações usamos para melhorar o contraste de áreas escuras de imagens? Explique o funcionamento.

R = Transformação de Power-Law
A fórmula geral é:

$$s = c \cdot r^{\gamma}$$

- r = intensidade do pixel original (normalizada entre 0 e 1).
- s = intensidade transformada.
- c = constante de escala (geralmente 1).
- γ = parâmetro que controla a "curvatura".

Como melhora áreas escuras:

- Se γ <1, os valores baixos de r são amplificados, tornando áreas escuras mais visíveis, enquanto os valores altos permanecem relativamente inalterados.
- Se γ >1, a imagem escurece.

Exemplo prático:
Para uma imagem escura, usar γ=0.5 deixa os detalhes das sombras mais perceptíveis.

Transformação Logarítmica
A fórmula é:

$$s = c \cdot \log(1 + r)$$

- r = intensidade do pixel (normalizada).
- c = constante de escala.

Como funciona:

- Expande os valores baixos e comprime os valores altos.
- Útil quando muitos detalhes estão concentrados em regiões escuras.

Observação:
Para aplicar corretamente, o pixel deve estar normalizado em [0,1] antes da transformação.

Stretching Linear de Intensidade
A fórmula é:

$$s = \frac{r - r_{\min}}{r_{\max} - r_{\min}}$$

- rmin e rmax = mínimos e máximos da intensidade na imagem.

Como funciona:

- "Estica" o histograma da imagem para ocupar todo o intervalo [0,1] ou [0,255].
- Pode aumentar o contraste de áreas escuras, mas de forma uniforme, sem curvar os valores.

3.2 Implemente no arquivo `transforms.py` a função `gamma(img, lambda)`. Esta função deve ler uma imagem usando `read_img()` e retornar a imagem transformada e normalizado. Teste a função usando a imagem acromáticas de sua escolha e realize o salvamento com `save_img()` definidas no arquivo common.py. Imprima e discuta os resultados em seu relatório.

Respondida no notebook

3.3 Matematicamente, qual é a função de transformação de intensidade que usamos para gerar negativos de imagens? Explique o seu funcionamento.

Seja rrr a intensidade original do pixel (normalizada entre 0 e 1) e L o valor máximo da escala (por exemplo, L=1 para imagens normalizadas ou L=255 para imagens em 8 bits), a transformação é:

$$s = L - r$$

ou, para imagens normalizadas:

$$s = 1 - r$$

- r = intensidade do pixel original
- s = intensidade do pixel transformado (negativo)

Funcionamento

1. Inversão de tons:
    - Pixels claros (r ≈ L) tornam-se escuros (s ≈ 0).
    - Pixels escuros (r ≈ 0) tornam-se claros (s ≈ L).
2. Visualização:
    - Desta forma, áreas que antes eram escuras aparecem claras e vice-versa, criando o efeito de "negativo fotográfico".
3. Aplicações:
    - Realce de detalhes em regiões escuras.

- Preparação de imagens para certas operações de processamento (ex.: detecção de bordas).

3.4 Implemente no arquivo `transforms.py` a função `negative(img)`. Esta função deve ler uma imagem usando `read_img()` e retornar o seu negativo normalizado. Teste a função usando a imagem `lenacolor.jpg` e realize o salvamento com `save_img()` definidas no arquivo **common.py**. Imprima e discuta os resultados em seu relatório.

Respondida no notebook

3.5 Por que correções de brilho no espaço RGB podem ser problemáticas? Qual o espaço de cor mais apropriado?

Problemas no espaço RGB

- **Mistura de luminância e crominância:**
  Cada canal (R, G, B) contém informação de **cor e brilho** ao mesmo tempo. Aumentar ou diminuir os valores de R, G e B proporcionalmente pode alterar não apenas o brilho, mas também a **saturação e a tonalidade** da cor.
- **Clipping e saturação de cores:**
  Ao aumentar o brilho, alguns canais podem atingir o valor máximo (255 ou 1), enquanto outros não, resultando em cores distorcidas. Por exemplo, uma cor azul clara pode virar roxo se o vermelho subir demais.
- **Não linearidade perceptual:**
  O espaço RGB não corresponde linearmente à percepção humana de brilho. Pequenas alterações podem ser pouco perceptíveis em tons escuros ou exageradas em tons claros.

Espaço de cor mais apropriado

Para manipulação de brilho e contraste, é melhor separar **luminância (intensidade)** da **crominância (cor)**. Os espaços de cor recomendados incluem:

- **HSV (Hue, Saturation, Value):**
  - O canal **V** representa o brilho (value).
  - Ajustar apenas V altera o brilho sem mudar tonalidade ou saturação.
- **HLS / HSL (Hue, Lightness, Saturation):**
  - O canal **L** representa a luminosidade.
  - Permite ajustar a luminosidade mantendo a cor (hue) constante.
- **YCbCr / YUV:**

- O canal Y representa luminância.
- Canais Cb e Cr contêm a informação de cor.
- Muito usado em vídeo e compressão de imagem, facilita ajustes de brilho sem afetar cores.

3.6 Como podemos interpretar o histograma de uma imagem em tons de cinza? Explique quando podemos inferir que uma imagem tem baixo contraste observando seu histograma.

O histograma de uma imagem em tons de cinza é uma representação gráfica da distribuição das intensidades de pixels da imagem. Ele mostra, para cada valor de intensidade rrr (0 a 255 em imagens de 8 bits), quantos pixels possuem aquela intensidade.

Como interpretar

- **Eixo x:** valores de intensidade (0 = preto, 255 = branco).
- **Eixo y:** quantidade de pixels com aquela intensidade.
- **Picos:** regiões com muitos pixels naquele nível de cinza.
- **Distribuição:** mostra se a imagem é clara, escura ou equilibrada:
  - Se o histograma está concentrado à esquerda → imagem escura.
  - Se está concentrado à direita → imagem clara.
  - Se ocupa toda a faixa → boa distribuição de tons, possivelmente alto contraste.

3.6.1 **Baixo contraste**

- Um **histograma estreito** (concentrado em uma pequena faixa de intensidades) indica **baixo contraste**.
- Características:
  - A maior parte dos pixels tem valores próximos entre si.
  - Detalhes em sombras ou regiões claras podem estar “achatados” e pouco perceptíveis.
- Exemplo visual:

- Uma imagem cinza uniforme com pequenas variações terá histograma centralizado, estreito, longe das extremidades (0 e 255).

3.7 Qual a finalidade da especificação de histograma. Explique matematicamente o procedimento de transformação da imagem de entrada.

Especificação de histograma é a transformação de uma imagem de modo que seu novo histograma corresponda a um histograma pré-especificado oriundo, por exemplo, de outra imagem.

- Aplicável quando se deseja que uma imagem tenha aparência semelhante a outra ou atenda a um padrão específico de tons.
- Controlar o **contraste e a distribuição tonal** da imagem de forma mais precisa do que apenas o alongamento linear (histogram equalization).
- Útil em áreas como **análise médica, visão computacional e fotografia**, onde a uniformidade tonal é importante

Obter a CDF escalada ($S\,r$ = (L − 1) ∑z $pr(j)$) da imagem de entrada I e a CDF escalada (G(z) = ∑z $pz(j)$) do histograma especificado.
Criar um mapeamento dos valores r dos pixels da imagem de entrada para valores z, de modo que a CDF de r tenha o mesmo percentil da CDF especificada para z. q A ideia é mapear cada valor de r em I para o valor z tal que S (r) = G (z) ou z = G^-1(S (r))
Obter a imagem de saída atribuindo aos pixels correspondentes da imagem de entrada seu valor de mapeamento z.

3.8 Qual a finalidade da equalização de histogramas? Qual é a função de transformação de intensidade envolvida? Explique o procedimento.

A equalização de histograma é usada para **melhorar o contraste global** de uma imagem.

- Redistribui as intensidades de forma que os níveis de cinza sejam usados **mais uniformemente**.
- Isso realça detalhes em regiões muito escuras ou muito claras.
- É muito útil quando a imagem tem **baixo contraste** (histograma concentrado em uma faixa estreita).

Função de Transformação de Intensidade

Seja:

- r = intensidade de entrada normalizada (0 ≤ r ≤ 1)
- s = intensidade de saída (transformada)

- pr(r) = distribuição de probabilidade da intensidade (histograma normalizado)

$$s = T(r) = \int_0^r p_r(w)\, dw$$

A função de transformação é dada pela **função de distribuição acumulada (CDF):**

- T(r) é **monótona crescente** e mapeia r para s ∈[0,1].
- Como consequência, a imagem resultante tem histograma aproximadamente **uniforme**.

Procedimento da Equalização

1. Calcular o histograma normalizado da imagem:

$$p_r(r_k) = \frac{n_k}{n}$$

onde nk é o número de pixels com intensidade rk e n é o total de pixels.

2. Obter a função de distribuição acumulada (CDF):

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j)$$

3. Aplicar a transformação:

Cada pixel da imagem original rk é mapeado para um novo valor sk.

4. Resultado:

O histograma da imagem transformada tende a ser uniforme, utilizando melhor toda a faixa de intensidades.

3.9 No arquivo `transforms.py` Implemente a função `histeq(img)`. Essa função recebe uma imagem como entrada e retorna a imagem equalizada. Se a imagem for colorida a equalização deve ser feita apenas na intensidade no espaço HSI. Para esse exercício é proibido usar **funções numpy** ou **Opencv** integradas para calcular histogramas ou realizar a equalização. Para fins de teste, utilize duas imagens de sua escolha, uma colorida e uma em escala de cinza, ambas com baixo contraste. Compare os resultados da sua implementação com os obtidos com `cv2.equalizeHist()`, plotando-os lado a lado.

Respondida no notebook

3.10 Explique o procedimento para realizar a especificação de histogramas de imagens.

Fazer com que a distribuição de intensidade de uma imagem de entrada (imagem original) se aproxime da distribuição de intensidade de uma imagem de referência.

Passos do procedimento

1. Calcular o histograma e CDF da imagem original
    - Contar quantos pixels existem em cada nível de intensidade.
    - Calcular a CDF (Cumulative Distribution Function) da imagem original.
2. Calcular o histograma e CDF da imagem de referência
    - Mesma coisa: contar pixels de cada nível e calcular a CDF da imagem de referência.
3. Criar a função de mapeamento
    - Para cada intensidade r da imagem original, encontrar a intensidade z na imagem de referência cujo CDF seja mais próximo do CDF de r.
    - Em outras palavras, encontrar zzz tal que:

$$\text{CDForiginal}(r) \approx \text{CDFreferência}(z)$$

4. Aplicar o mapeamento à imagem original
    - Substituir cada pixel r da imagem original pelo valor correspondente z obtido na etapa anterior.

4 Filtragem Espacial.

4.1 **Os filtros espaciais gaussianos 1D e 2D são derivados das funções gaussianas 1D e 2D especificadas abaixo. Para um filtro gaussiano 2D com uma dada variância σ^2, a convolução pode ser reduzida por operações sequenciais de um kernel 1D. Prove que uma convolução por um filtro Gaussiano 2D é equivalente a convoluções sequenciais de um filtro Gaussiano 1D vertical e horizontal. Especifique a relação entre o filtro gaussiano 2D e 1D, especialmente a relação entre suas variâncias.**

1D kernel : $G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$ 2D kernel : $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$

Observe que o gaussiano 2D pode ser fatorado:

$$G_2(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \underbrace{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}}_{G_1(x)} \cdot \underbrace{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}}_{G_1(y)}$$

Ou seja:

$$\boxed{G_2(x,y) = G_1(x) \cdot G_1(y)}$$

Convoluir uma imagem com um gaussiano 2D é equivalente a convoluir primeiro com um gaussiano 1D horizontal e depois vertical (ou vice-versa)

4.2 Implemente a função `convolve(img, kernel2d, padd=False)` no arquivo `filter.py` que realiza a convolução da imagem com o kernel 2D fornecido. Compare seus resultados com os obtidos com `cv2.filter2d()`

Respondida no notebook

4.3 Teste a implementação da função `convolve()` com a imagem `gracehopper.png` e um kernel gaussiano 3 × 3. Plote as imagens de saída em seu relatório. Descreva de forma sucinta o que a filtragem gaussiana fez com a imagem.

Respondida no notebook

4.4 Considere imagens como funções $I(x,y) : R^2 \rightarrow R$. Quando trabalhamos com detecção de bordas (veremos a frente no curso), precisamos prestar atenção às aproximações das derivadas:

$$I_x(x,y) = \frac{\partial I}{\partial x}(x,y) \approx \frac{1}{2}(I(x+1,y) - I(x-1,y))$$
$$I_y(x,y) = \frac{\partial I}{\partial y}(x,y) \approx \frac{1}{2}(I(x,y+1) - I(x,y-1))$$

Construa kernels de convolução com base nas aproximações acima: (i) $k_x \in R^{3\times1}$, (i) $k_y \in R^{1\times3}$. Implemente a função **edgedetection(img)** no arquivo `filter.py`. Ele deve retornar a magnitude do gradiente. As componentes devem ser calculadas com os kernels k_x e k_y. Teste a função **edgedetection()** com a imagem **lenagray.jpg**, salve os resultados e os discuta em seu relatório.

Respondida no notebook

4.5 Os filtros de Sobel computam uma aproximação para a derivadade uma imagem. Implemente a função **sobel(img)** no arquivo **filters.py.** Teste a função **sobel()** com a imagem **lenagray.jpg** salve os resultados e discuta em seu relatório. Compare com os resultados obtidos no exercício 4.4.

Respondida no notebook

4.6 Chamamos de *steerable filter* um operador que computa a derivada em qualquer direção.

(a) Revise o conceito de derivadas direcionais e derive um kernel que detecta transições em termos de um ângulo α, tomando como base o grandiente aproximado obtido pelos kernels de Sobel.

(b) Implemente a função `steerablefilter(img, alpha)` em **filters.py**. Teste sua função com $\alpha = 0, \frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}, \frac{2\pi}{3}, \frac{5\pi}{6}$ e a imagem **gracehopper.png**. Salve os resultados e discuta em seu relatório.

Respondida no notebook