Trabalho 1 de Introdução ao Processamento de Imagem Digital

Cauan Newton Alves Souza
RA 195620

1) Introdução

Esse trabalho tem como objetivo a realização de alguns processamentos básicos em imagens digitais. Como exercício de introdução aos conceitos da matéria, temos uma lista de transformações que foram implementadas em código a fim de entender as operações envolvidas no processamento de imagens. Nesse relatório, teremos uma descrição de como o código foi implementado, uma análise dos conceitos aplicados e, por fim, uma discussão dos resultados obtidos e das limitações observadas.

2) Implementação

O código implementado para realizar as transformações propostas se encontra no arquivo *lab1.py* que foi entregue conjuntamente a este documento. Ele foi escrito em Python3 e faz uso das seguintes bibliotecas:

i. *sys*: Para aquisição de argumentos passados na linha de comando durante a execução do código;
ii. *opencv-python*: Para operações básicas e fundamentais em processamento de imagem, como abrir, exibir e criar imagens, editar escala de cores, aplicar filtros a imagens, entre outras funções;
iii. *numpy*: Usada em operações matemáticas e manipulação de vetores.

A execução do código é feita pelo script lab1.py. Este recebe dois parâmetros de entrada: a imagem a ser processada e um número identificador de qual transformação deve ser aplicada. A relação entre o número identificador e a transformação realizada pode ser vista a seguir.

1	Mosaico
2	Combinação de Imagens
3	Negativo
4	Conversão de Intervalo
5	Inversão das Linhas Pares
6	Reflexão da Metade Superior
7	Espelhamento Vertical
8	Transformação de Cores
9	Transformação Monocromática
10	Correção Gama
11	Quantização
12	Plano de Bits
13	Filtragem
14	Filtragem (Caso Especial)

Como exemplo, podemos escrever a seguinte linha de comando no terminal para aplicar espelhamento vertical a uma imagem.

```
$ python lab1.py img/baboon.png 7
```

Ao ser executado, o programa abre a imagem endereçada, aplica a transformação desejada, apresenta o resultado e exporta a imagem em um novo arquivo nomeado *output.png*. Na pasta *examples* é possível ver imagens onde aplicamos essas transformações.

3) Análise do Código

O primeiro padrão de código que podemos notar em todas as transformações é a manipulação de pixels da imagem. A aplicação dos efeitos ocorre pela iteração de laços que percorrem cada pixel da imagem e realizam uma operação matemática, seja uma soma com outra imagem, uma multiplicação por escalar, vetor ou matriz, uma conversão de escala, entre outras transformações possíveis.

O que torna essa observação relevante é que operações iterativas são computacionalmente custosas. Um efeito aplicado a todo pixel de uma imagem terá tempo de execução na ordem de $O(n^2)$, o que indica um crescimento quadrático no tempo de processamento conforme a resolução de uma imagem cresce. Por isso, é preferível que muitas dessas operações sejam feitas de forma vetorizada. As bibliotecas *numpy* e *openCV* dispõem de recursos que permitem tanto a vetorização quanto a otimização de comandos aplicados a imagens.

Como exemplo, tomemos a transformação 2, onde fazemos uma combinação entre duas imagens. Temos duas formas de executar esse processo: podemos fazer uma soma iterativa entre os pixels de cada imagem, ou podemos usar uma função pronta da biblioteca OpenCV para fazer uma soma ponderada entre imagens. Comparando o tempo de execução entre essas duas formas, temos uma diferença de tempo na escala dos milhares.

```
bash-5.2$ python test2.py
Time using iteration 0.997383824 s
Time using OpenCV function: 9.9611e-05 s
Ratio: 10012.787985262672
```

Figura 1: Diferenças de tempo entre uma transformação por método iterativo e por função pronta da biblioteca.

Para a resolução dos casos propostos no trabalho, o tempo de execução otimizado não foi uma prioridade. Para uma boa parte da implementação foram escritos algoritmos que fossem suficientes em aplicar a transformação exigida, e por isso temos um forte uso de soluções iterativas.

Outro detalhe que torna a implementação mais simples é que a maioria das transformações são aplicadas a imagens monocromáticas. Os algoritmos aplicados a imagens assim são mais intuitivos de escrever e de corrigir, além de serem computacionalmente mais leves se comparados a algoritmos aplicados a imagens coloridas.

Entre as transformações aplicadas, talvez a mais interessante seja a filtragem de imagens. Por meio de uma matriz, chamada de máscara, podemos aplicar uma transformação a uma imagem com base na vizinhança de cada pixel. Ao aplicar a máscara, fazemos uso da convolução para calcular o novo valor do pixel transformado. O que torna máscaras úteis é que elas são fáceis de representar (basta uma matriz), aplicam transformações bem relevantes pra área de imagem e geralmente são fáceis de operar, já que muitas bibliotecas de imagem tem suporte para a aplicação de máscaras. A seguir, podemos ver alguns dos filtros explorados nesse trabalho.

1	Filtro Passa-Altas
2	Filtro Gaussiano
3	Detector de Borda Vertical
4	Detector de Borda Horizontal
5	Filtro Passa-Altas
6	Filtro-Caixa
7	Filtro Passa-Altas

8	Filtro Passa-Altas
9	Filtro Borrador
10	Filtro Passa-Banda
11	Detector de Borda

Uma observação interessante é que a combinação dos filtros 3 e 4 resulta em um detector de bordas gerais, o que foi verificado experimentalmente no caso especial (transformação 14) desse trabalho. Isso evidencia a facilidade de fazer operações com máscaras, mesmo que combinadas, quando estamos filtrando imagens.

4) Resultados e Limitações

Muitos dos exercícios propostos foram resolvidos, mas não todos. Em particular, os exercícios de mosaico e de planos de bits se mostraram bem difíceis de implementar, por motivos diferentes. Para a construção do mosaico, não é difícil projetar um algoritmo capaz de fazer a transformação proposta, mas o código não foi completo por erros no processamento de diferentes tipos de dados por funções da biblioteca OpenCV. Isso mostra que, mesmo com a conveniência de ter bibliotecas prontas para processar imagens, a implementação do código não é tarefa trivial.

Como dito, também foi difícil implementar a transformação de planos de bits. Ainda que conceitualmente simples de entender, sua implementação é difícil por envolver operações matemáticas bem específicas. A criação de um algoritmo que resolva esse caso depende de um bom conhecimento das ferramentas matemáticas e de manipulação de bits disponíveis na linguagem Python.

Ainda assim, a entrega parcial ainda se mostrou relevante para o entendimento da matéria. A maioria dos exercícios foram resolvidos e isso permitiu um entendimento prático dos conceitos estudados em Processamento de Imagens.

5) Bibliografia

OpenCV-Python Tutorials. Disponível em <https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html>. Acesso em 5 de abril de 2023.