

# Introdução ao Processamento de Imagem Digital

## Trabalho 5

RAYSA MASSON BENATTI

RA 176483

### I. ESPECIFICAÇÃO DO PROBLEMA

O presente relatório apresenta a descrição das conclusões e estratégias adotadas para a resolução do Trabalho 5 da disciplina Introdução ao Processamento de Imagem Digital (MC906/MO443), ministrada no primeiro semestre de 2019 na Unicamp pelo professor Hélio Pedrini.

O trabalho consistiu em aplicar alguma técnica de agrupamento de dados (*clusterização*) — ou seja, aprendizado de máquina não supervisionado — para encontrar grupos de cores mais representativas de uma imagem e, com isso, reduzir (quantizar) seu número de cores, procurando manter, na medida do possível, a qualidade global de sua aparência.

Foi aplicada a técnica *k-means*, com diferentes valores de  $k$ .

### II. ENTRADA DE DADOS

A imagem de entrada é sempre uma imagem colorida no formato PNG (*Portable Network Graphics*). Os testes foram realizados com duas imagens distintas, "peppers.png" e "baboon.png", ambas retiradas de [1].

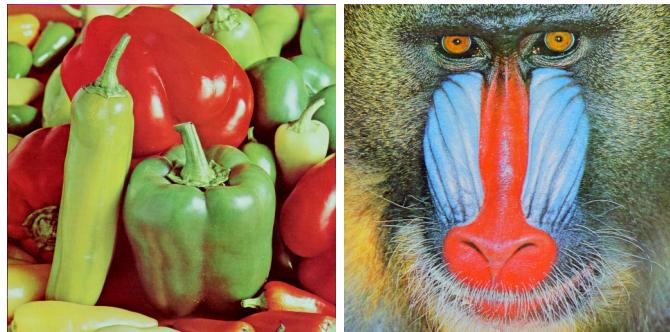


Figura 1. Imagens originais sobre as quais a técnica foi aplicada.

Para a leitura das imagens, foi utilizado o método `imread(filename)`, que faz parte da biblioteca OpenCV. A imagem é sempre lida no modo *default*, sem nenhum parâmetro adicional, e armazenada em uma variável sobre a qual são feitas, em seguida, as manipulações necessárias.

### III. SAÍDA DE DADOS

A imagem resultante das operações é recuperada com o método `imwrite(filename, img)` da biblioteca OpenCV, sendo gravada também no formato PNG.

### IV. SOLUÇÃO

A implementação da técnica de *k-means* foi adotada conforme descrito em [2]. Inicialmente, a imagem é redimensionada para uma matriz de tamanho  $M \times 3$ , em que  $M$  é o número total de pixels, e seus valores são convertidos para o tipo `float32`. Isso porque há três *features* relevantes na imagem: R (cor vermelha), G (cor verde) e B (cor azul), cada uma das quais deve ser colocada em uma coluna, para que o método `cv2.kmeans()` funcione.

Em seguida, são definidos os critérios de parada do algoritmo. Trata-se de uma tupla de três valores:

- **Tipo:** define o tipo de critério de parada utilizado. Há três *flags* possíveis, das quais usamos duas: `cv2.TERM_CRITERIA_EPS` (atingir determinado valor de acurácia *epsilon*) e `cv2.TERM_CRITERIA_MAX_ITER` (atingir número máximo de iterações);
- **Iterações:** define o número máximo de iterações — 10 no nosso caso;
- **Epsilon:** define o valor da acurácia desejado — 1.0 no nosso caso.

O valor de  $k$  é, então, definido em uma variável. Na solução proposta, testamos valores de  $k = 16, 32, 64$  e  $128$ . Esse valor define a quantidade de grupos (*clusters*) desejados.

O passo seguinte é efetivamente invocar o método `cv2.kmeans`, o qual recebe cinco parâmetros: a imagem redimensionada, isto é, as amostras que estarão sujeitas ao agrupamento; o valor de  $k$ ; os critérios de parada definidos anteriormente; o número de tentativas de execução do algoritmo com rótulos iniciais diferentes — que definimos como 10; *flags* que especificam como os centroides iniciais são definidos — no nosso caso, optamos por ativar `cv2.KMEANS_RANDOM_CENTERS`.

A função retorna três valores: `ret` armazena a compactação, isto é, a soma da distância ao quadrado de cada

ponto para seus respectivos centroides; `label` armazena o vetor de rótulos correspondentes a cada pixel da imagem (*codebook*); `center` armazena os centroides de cada grupo.

Para recuperar a imagem resultante, os valores dos centroides são convertidos de volta para o tipo `uint8` e indexados pelos valores dos rótulos colapsados em apenas uma dimensão com o método `flatten`. O resultado é usado para construir a imagem final com o retorno às dimensões originais (`reshape`).

O tempo de execução do método `cv2.kmeans` é medido com auxílio do método `time.time()`, a fim de comparar o desempenho do algoritmo para diferentes valores de  $k$ .

## V. RESULTADOS E DISCUSSÕES

Para a imagem "*peppers.png*", os resultados exibidos foram os seguintes:



Figura 2. Resultados da aplicação da técnica  $k$ -means sobre a imagem dos pimentões. Da esquerda para a direita, de cima para baixo:  $k = 16, 32, 64$  e  $128$ .

Os tempos de execução, medidos em segundos e arredondados para duas casas decimais, foram de  $0.86$ ,  $1.60$ ,  $2.81$  e  $5.26$ , para os valores de  $k = 16, 32, 64$  e  $128$ , respectivamente.

Para a imagem "*baboon.png*", os resultados obtidos foram os seguintes:

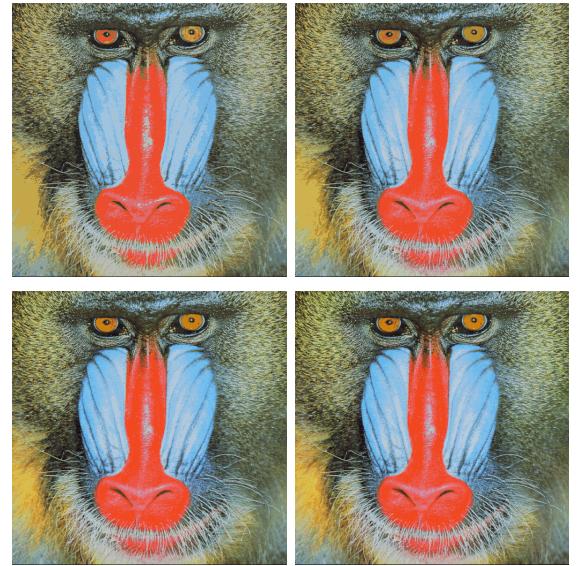


Figura 3. Resultados da aplicação da técnica  $k$ -means sobre a imagem do babuíno. Da esquerda para a direita, de cima para baixo:  $k = 16, 32, 64$  e  $128$ .

Os tempos de execução, medidos em segundos e arredondados para duas casas decimais, foram de  $0.94$ ,  $1.66$ ,  $2.97$  e  $5.53$ , para os valores de  $k = 16, 32, 64$  e  $128$ , respectivamente.

Os resultados obtidos são coerentes com o que se espera do algoritmo, que, de fato, atinge o objetivo de agrupar diferentes cores da imagem e exibi-la de acordo com esse agrupamento. Quanto menor o valor escolhido de *clusters*, mais informação é retirada da imagem original; ao aumentar o valor de  $k$ , a imagem se aproxima da aparência que tem com o esquema de cores completo. Essa diferença é mais evidente na imagem dos pimentões — embora seja possível observá-la também nos detalhes da outra imagem, como a cor dos olhos do babuíno, por exemplo.

Os tempos de execução obtidos, também previsivelmente, aumentam conforme o número de grupos a definir aumenta, de maneira similar nas duas imagens. A tendência de crescimento segue a proporção do aumento de  $k$ : quando este é dobrado, também aproximadamente dobra o tempo gasto para execução, sugerindo se tratar de algoritmo de complexidade linear em função do tamanho da entrada.

## REFERÊNCIAS

- [1] Hélio Pedrini. Banco de imagens disponível em [http://www.ic.unicamp.br/~helio/imagens\\_coloridas/](http://www.ic.unicamp.br/~helio/imagens_coloridas/).
- [2] *K-Means Clustering in OpenCV*. Disponível em <https://bit.ly/2WT6phL>.