SCC0275 - Introdução à Ciência de Dados
Primeiro Projeto Prático

Neste projeto vamos comparar de maneira mais aprofundada duas estratégias para **quantizar** imagens. Uma delas é a quantização uniforme e outra usa o algoritmo de agrupamento K-Means. Além disso, vamos entender melhor o papel que uma boa métrica de erro cumpre na hora de comparar soluções e determinar sua qualidade.

Alguns códigos são fornecidos no arquivo **projeto1.ipynb**. Fique atento, pois esses códigos são mencionados nos enunciados das questões. Utilize a biblioteca scikit-learn para criar os modelos pedidos. Você pode usar outras bibliotecas para fazer as demais análises =)

Todas as respostas devem ser justificadas com base em:

1. **Código Python mostrando a(s) análise(s) e/ou o(s) modelo(s) feitos;**
2. **O resultado da(s) análise(s) e/ou do(s) modelo(s) e**
3. **Uma explicação textual (pode ser breve) da conclusão obtida.**

Em caso de plágio (mesmo que parcial) o trabalho de todos os alunos envolvidos receberá nota ZERO.

NÃO enviar um link para o Google Colab como resposta/relatório do projeto.

Desorganização excessiva do código resultará em redução da nota do projeto. Exemplos:

- **Códigos que devem ser rodados de forma não sequencial;**
- **Projeto entregue em vários arquivos sem um README;**
- **...**

Bom projeto,
Tiago.

Introdução

O Notebook disponibilizado tem uma classe chamada **MyImgFormat**, que permite que uma imagem RGB seja armazenada utilizando menos bytes. Um objeto desta classe é construído com base em:

1. Uma matriz de IDs para cada pixel (chamada de **mat_rgb_ids** no construtor da classe) que assume que os IDs estão entre zero e N (número de cores da imagem);
2. Um dicionário (**rgb_ids_dict**) que mapeia cada ID para o RGB da cor. Como no exemplo a seguir:

```
{
    0: [178, 195, 203],
    1: [51 , 54 , 43 ],
    2: [225, 225, 227],
    3: [120, 112, 62 ]
}
```

Além disso, este notebook tem um função chamada **uniform_quant** que permite que uma imagem seja quantizada, dividindo os intervalos de valores R, G e B de forma uniforme. Para fazer isso, essa função usa dois inputs: uma imagem RGB (que deve conter número inteiros entre 0 e 255) e uma quantidade total de cores desejada (número inteiro) na imagem quantizada. Os valores retornados por essa função são: uma matriz de IDs e um dicionário que mapeia os IDs para RGBs. Ou seja, os dados necessários para criar um objeto do tipo **MyImgFormat**.

Finalmente, temos a função **get_bin_size_kb** que calcula a quantidade de KBs usadas por um objeto em um arquivo. Com ela poderemos estimar o tamanho ocupado por imagens no disco.

Questões

Questão 1 (valor 2,5 pontos)

a) Ao estudar a função **uniform_quant** você irá perceber que ela gera os valores de R, G e B como o valor inicial do intervalo quantizado. Por exemplo, ao quantizar uma imagem uma parte do dicionário obtido é o exibido na figura abaixo. Modifique a função para que os valores de RGB sejam o meio do intervalo quantizado.

```
{
    0: [0  , 0  , 0  ],
    1: [0  , 0  , 128],
    2: [0  , 128, 0  ],
    3: [0  , 128, 128],
    4: [128, 0  , 0  ],
    5: [128, 0  , 128],
    6: [128, 128, 0  ],
    7: [128, 128, 128]
```

b) Carregue a imagem **'china.jpg'** usando o scikit-learn e mostre (no Jupyter Notebook) as 3 versões da imagem:
   - Original;
   - Quantizada para 64 cores usando o código original;
   - Quantizada para 64 cores usando o código feito no item anterior.

c) Comparar tamanho em KB das 3 imagens da questão anterior. (Dica: use a função **get_bin_size_kb**)

Questão 2 (valor 2,5 pontos)

a) Implementar uma estratégia de quantização de imagens usando o K-Means para determinar os valores RGB de cada um dos IDs das cores. Modifique os valores dos parâmetros do construtor da classe K-Means como achar necessário, mas deixe os parâmetros listados abaixo com os valores indicados para que seja fácil reproduzir seus resultados:
    - max_iter=10,
    - random_state=42

b) Rode o método usando o K-Means para quantizar a imagem **'china.jpg'** para 64 cores e compare visualmente a imagem gerada com os resultados da Questão 1. Comente os resultados.

Questão 3 (valor 2,5 pontos)

a) Existem diversas maneiras de se estimar a quantidade "ótima" de clusters para um algoritmos como K-Means. Uma delas é uma análise baseada na medida de inércia, que é apresentada neste link. Implemente o cálculo da inércia dentro da função de quantização baseada no K-Means.

b) Implemente uma função de comparação que calcula o MSE entre os pixels da imagem original e os da imagem quantizada.

Questão 4 (valor 2,5 pontos)

Responda os itens a seguir usando as imagens **'china.jpg'** e **'flower.jpg'** do scikit-learn.

a) Faça a quantização das imagens para {8, 27, 64, 125, 216} cores usando ambos os métodos de quantização (uniforme e K-Means)

b) Compute e faça um plot da inércia dos resultados das quantizações do K-Means obtidas no item a)

c) Compute e faça um plot dos MSEs de ambas as estratégias. Qual das estratégias teve um melhor MSE? Isso faz sentido quando fazemos uma comparação visual dos resultados?

d) Compute e faça um plot do tamanho e KBs das imagens quantizadas e compare com o tamanho em KBs da imagem original. Faça a comparação considerando os seguintes pontos:
    - Use a função **get_bin_size_kb** para calcular os tamanhos das imagens em KBs;
    - Para a imagem original, calcule o tamanho do numpy.ndarray que contém os pixels RGB da imagem;
    - Para as imagens quantizadas, compute o tamanho do objeto MyImageFormat criado usando os resultados da quantização.