Projeto 3 de Fundamentos de Compressão de Sinais
Quantização

Giordano Süffert Monteiro - 17/0011160

I. Codificador CIQA

A. Descrição do algoritmo codificador

O codificador (ciqa_encoder.py), é baseado em uma quantização adaptativa e ,primeiramente, espera que seja informado a dimensão de cada bloco quadrado que será utilizado (N), o número de níveis do quantizador e a imagem em escala de cinza a ser quantizada.

A execução desse algoritmo pode ser resumida nos seguintes passos:

1) Padding da imagem: Como o próprio nome já diz, é realizado um preenchimento na imagem de forma a adicionar colunas ou linhas de valores 0 a fim de tornar as dimensões da imagem divisíveis pelas dimensões do bloco (N).

2) Extração dos blocos: Para a extração dos blocos, é considerado um quadrado de lados de N pixels e a suposição de que a imagem pode ser divida em um número inteiro desses blocos. Dessa forma, são extraídos os pixels referentes a cada bloco e agrupados deformando sua estrutura original para uma única lista.

3) Quantização dos blocos: Tendo em mão todos os blocos devidamente separados, para cada um desses blocos, é realizada um quantização uniforme porém utilizando como os possíveis valores aqueles entre os máximos e mínimos locais, ou seja, supondo que em cada bloco existe uma distribuição uniforme das cores com o seu próprio mínimo e máximo.

4) Escrita no arquivo: Após a realização de todas as operações necessárias, resta somente colocar tudo no arquivo. Como informação laterial temos os primeiros 4 bits (0xF), seguido por 4 bits indicando o padding final do arquivo, além de N, M, número de blocos na horizontal da imagem, padding de linhas e padding de colunas, cada um em um byte. Em seguida, são escritos, para cada bloco o valor mínimo e máximo em um byte cada e os índices resultantes da quantização utilizando o número mínimo de bits necessário para representar M índices diferentes.

B. Descrição do algoritmo decodificador

Já decodificador (ciqa_openner), apresenta como função decodificar as informações existentes no arquivo resultante da codificação e mostrar a imagem decodificada para o usuário. Para realizar tais funções, primeiramente ele extrai as informações laterais informadas no arquivo, seguido pela extração da informação codificada.

1) Decodificação: De posse da informação lateral, para cada conjunto/bloco são extraídos o valor mínimo e máximo, seguido de N*N indices. Assim, foram reconstruídos os valores de reconstrução e substituídos cada índice pelo seu valor indicado.

2) Reconstrução da imagem: Tendo todos os novos valores, cada bloco é colocado no tamanho NxN e concatenado com os outros de forma a respeitar o número de blocos por linha. Com isso, a imagem resultante é exibida ao usuário. Exemplos mostrando a imagem original e a imagem resultante podem ser visualizados na figura 1.

II. Codificador CIVQ

A. Descrição do algoritmo criador de codebook

Esse programa (civq_codebook.py) tem como função gerar um arquivo com um codebook, o tamanho do codebook (M), o tamanho dos blocos a serem considerados (L) e a imagem em escala de cinza a ser quantizada.

Os dois primeiros passos na execução desse algoritmo são iguais àqueles do codificador CIQA, Padding da imagem e Extração dos blocos, em que L = N*N. Após essas etapas, temos as seguintes:

1) Criação dos code vectors: De posse de todos os blocos/elementos da imagem como uma sequência de pixels, é executado o algoritmo *kmeans* para gerar M clusters em cima utilizando esses elementos, informando os code vectors como os centros desses clusters.

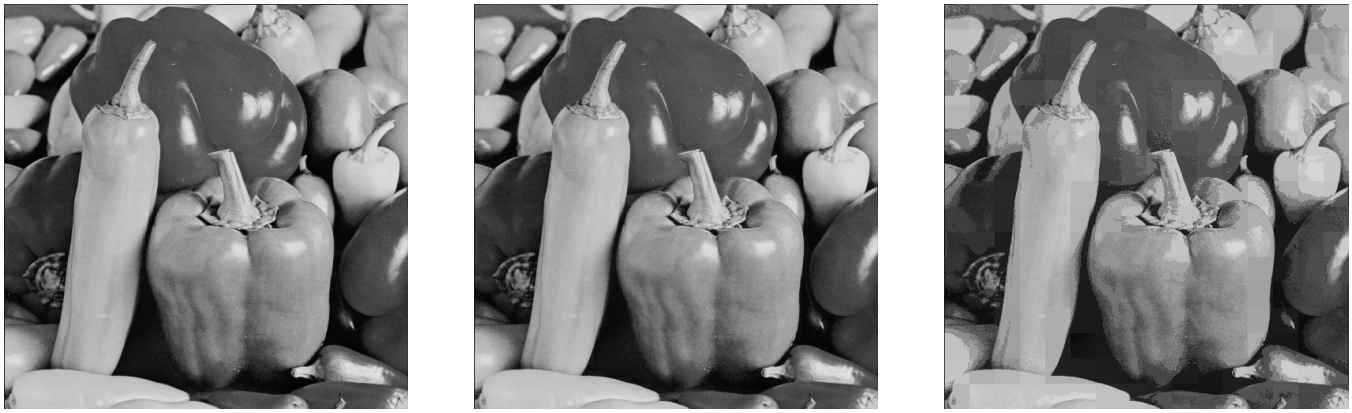
2) Escrita no arquivo .cdb: Para a escrita no arquivo (.cdb), a informação lateral é composta somente pelos valorres de M-1 (1 a 256) e L, ocupando um byte para cada. Em seguida, são escritos os M code vectors em sequência, utilizando um byte para cada um de seus L pixels.

B. Descrição do algoritmo codificador

O codificador (civq_encoder.py), baseado na na codificação vetorial, recebe um arquivo com a imagem em escala de cinza a ser codificada e outro com o codebook a ser utilizado nesse codificação (.cdb). Seu comportamento pode ser divido nas seguintes etapas:

1) Leitura do codebook: O arquivo contendo o codebook é lido e fornece todos os code vectors, além dos valores de M e L.

2) Codificação da imagem: Simularmente ao gerador de codebook, primeiro são feitos os passos Padding da imagem e Extração dos blocos do codificador CIQA. Em seguida, para cada bloco é calculado o índice do elemento do codebook que mais se aproxima de seu valor, criando uma lista com esses índices para a imagem inteira.



(a) Imagem original (b) Quantizada N=8 M=8 (c) Quantizada N=32 M=4

Fig. 1: Imagem original *peppers* e resultados do CIQA

3) Escrita no arquivo .cvq: Após a realização de todas as operações necessárias, resta somente colocar tudo no arquivo. Primeiro temos os primeiros 4 bits 1 mais 4 bits indicando o padding final do arquivo. Além disso, o arquivo do codebook é reescrito de forma integral nesse novo arquivo (M, L, code vectors). Em seguida, são escritas informações laterias para ajudar na reconstrução: número de blocos na horizontal, linhas de padding e colunas de padding, sendo um byte para cada. Assim, falta somente escrever a sequência de índices codificados, cada um utilizando o número mínimo de bits necessário para representar M índices diferentes.

C. Descrição do algoritmo decodificador

Já decodificador (civq_openner.py), apresenta como função decodificar as informações existentes no arquivo resultante da codificação e mostrar a imagem decodificada para o usuário. Para realizar tais funções, primeiramente ele extrai as informações do codebook, seguido por outras informações laterais e a informação codificada.

1) Decodificação: De posse da informação lateral e do codebook, para cada índice, seu valor é substituído pelo valores de reconstrução de mesmo índice no codebook reconstruído.

2) Reconstrução da imagem: Tendo todos os novos valores, cada bloco é colocado no tamanho L e concatenado com os outros de forma a respeitar o número de blocos por linha. Com isso, a imagem resultante é exibida ao usuário. Exemplos mostrando a imagem original e a imagem resultante podem ser visualizados na figura 2.

III. Codificador CIMap

A. Descrição do algoritmo codificador

O codificador (cimap_encoder.py), baseado também na codificação vetorial, espera receber o número de cores a serem utilizadas (M) e o nome da imagem colorida a ser codificada. Seu comportamento pode ser resumido nas seguintes etapas:

1) Geração do codebook e Quantização: Considerando cada pixel da imagem como 3 valores de 8 bits, e que um elemento/bloco é um pixel, foi utilizado o algoritmo *kmeans* sobre todos os pixels da imagem, gerando M valores centrais dos clusters e uma lista com o índice do cluster a qual um pixel pertence para todos os pixels da imagem, ou seja, seus valores quantizados.

2) Escrita no arquivo .cmp: Primeiramente são escritos 4 bits 1 e 4 bits informando o padding final do arquivo. Em seguida, foram escritas seguintes informações laterais: 8 bits para M-1 e 2 bytes para o número de colunas de pixels na imagem. Assim, são escritos os índices dos pixels codificados, cada um utilizando o número mínimo de bits necessário para representar M índices diferentes.

B. Descrição do algoritmo decodificador

Já decodificador (cimap_openner.py), apresenta como função decodificar as informações existentes no arquivo resultante da codificação e mostrar a imagem decodificada para o usuário. Para realizar tais funções, primeiramente ele extrai as informações laterais (M+1, número de colunas), seguido por informações do codebook e a informação/índices codificada.

O passo de decodificação é o mesmo do codificador CIVQ. Já a reconstrução da imagem, é realizada concatenando cada pixel/bloco respeitando a dimensão horizontal da imagem. Assim, a imagem resultante é exibida ao usuário. Exemplos mostrando a imagem original e a imagem resultante podem ser visualizados na figura 3.

IV. Dithering

A. Descrição do algoritmo codificador

O codificador (dit_encoder.py), espera receber o número de níveis do quantizador (M), nome da imagem a ser codificada e se deve ser realizado o dithering.

Para a quantização foi realizada uma quantização uniforme de M níveis de 0 a 255, sendo realizada em cada byte. Caso o dithering seja ativado, os bytes, de mesma profundidade, no próximo pixel, no pixel abaixo, no pixel um anterior do abaixo e no próximo do abaixo, são adicionados por um fator (7/16, 3/16, 5/16, 1/16) vezes o erro de quantização do pixel base.

Na escrita do arquivo .dit, são escritos primeiramente 3 bits 1, 1 bit para indicar se a imagem é em escala de cinza (0) ou colorida (1) e em seguida 4 bits indicando o padding

(a) Imagem original (b) Quantizada L=16 M=32 (c) Quantizada L=4 M=32

Fig. 2: Imagem original *lena* e resultados do CIVQ

(a) Imagem original (b) Quantizada M=16 (c) Quantizada M=256

Fig. 3: Imagem original *kodim05* e resultados do CIMap

no final do arquivo. Em seguida, como informação lateral, é escrito 1 byte para M-1 e 2 bytes para o tamanho horizontal de pixel da imagem. Com isso, os bytes codificados são escritos utilizando o número mínimo de bits necessário para representar M índices diferentes.

B. Descrição do algoritmo decodificador

Já decodificador (dit_openner.py), apresenta como função decodificar as informações existentes no arquivo resultante da codificação e mostrar a imagem decodificada para o usuário. Para realizar tais funções, primeiramente ele extrai as informações laterais (M+1, número de colunas, se é colorida ou não) e a informação codificada.

1) Decodificação:

2) Reconstrução da imagem: Tendo todos os novos valores, byte é organizado de acordo com a profundidade necessária em cada pixel e concatenado com os outros de forma a respeitar o número de pixels por linha. Com isso, a imagem resultante é exibida ao usuário. Exemplos mostrando a imagem original, a imagem resultante sem dithering e com dithering podem ser visualizadas na figura 4.

V. Comparações e Conclusão

Para observarmos a eficiência dos algoritmos desenvolvidos, foram plotados gráficos (figura 5) MSE x BPP de forma a comparar resultados para diferentes parâmetros em cada um dos algoritmos. Adicionalmente, para os codificadores CIQA e CIQV, foram escolhidos pontos de forma a tentar traçar um estimado lower convex hull.

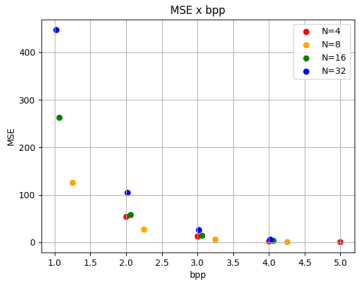
Além disso, afim de comparar a performance dos três primeiros quantizadores, as mesmas imagens foram comprimidas em .jgp e comparadas em um gráfico PSNR x BPP, como pode ser visto na figura ref.

Com esses resultados, é possível observar que todos os algoritmos comparados com o jpg foram piores em quase todos os momentos testado, com exceção do CIQV com poucos bpp. Já sobre as diferenças dos parâmetros para cada algoritmo, é possível observar que, no caso do CIQA, quanto maior o valor de M, menor a diferença que o tamanho do bloco (N) faz em relação à MSE. No caso do CIQV, mesmo com um relativo mesmo grande valor de M utilizar um pequeno L faz com que seu MSE seja baixo, porém também faz com que seja mais difícil ter um pequeno valor de bpp mesmo para menores M.

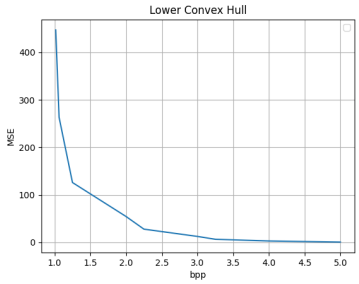
Em relação ao CIMap, o aumento de bpp e diminuição de MSE se comportam como o esperado com o aumento de M para um mesemo L. Já em relação ao dithering, é possível determinar que, tanto para imagens em tons de cinza quanto para a colorida, a que não utiliza dithering apresenta um maior PSNR como o esperado, sendo essa diferença aparemente a mesma para ambos os casos.

(a) Imagem original (b) Quantizada M=2 sem Dithering (c) Quantizada M=2 com Dithering

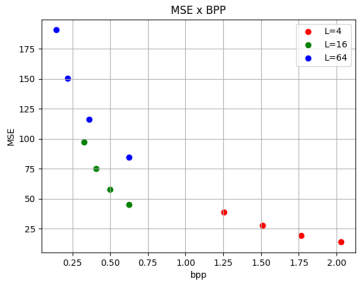
Fig. 4: Imagem original *lena* e resultados com e sem dithering



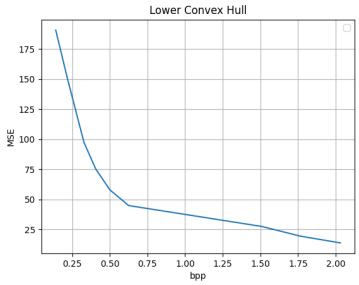
(a) CIQA utilizando peppers.bmp



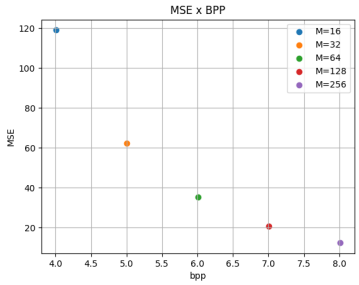
(b) CIQA Lower Convex Hull



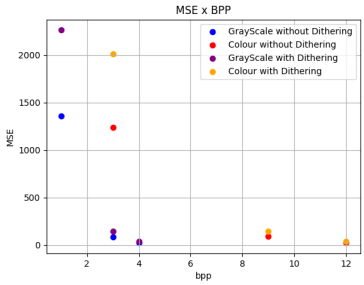
(c) CIQV utilizando lena.bmp



(d) CIQV Lower Convex Hull

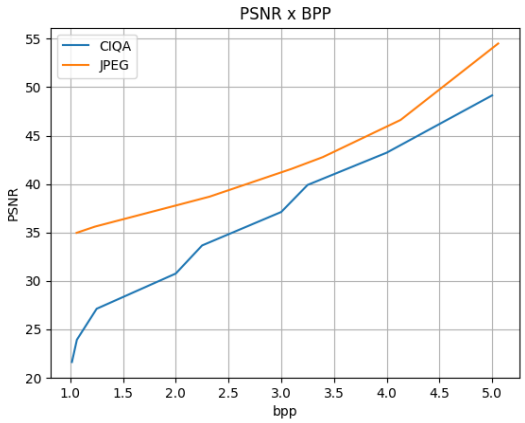


(e) CIMap com kodim05.png

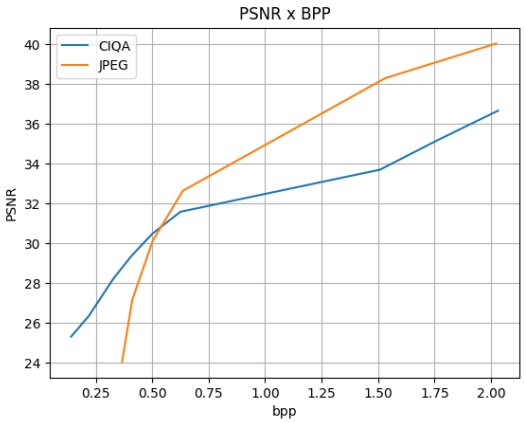


(f) Dithering lena e lena_colour

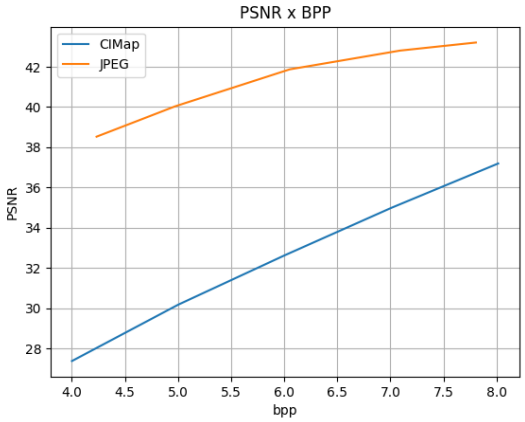
Fig. 5: MSE x BPP dos algoritmos desenvolvidos



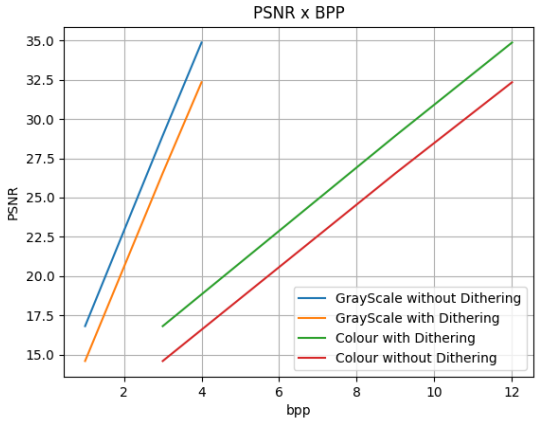
(a) CIQA x JPEG peppers



(b) CIQV x JPEG lena



(c) CIMap x JPEG kodim05



(d) Only quantization x Dithering

Fig. 6: PSNR x BPP dos algoritmos