INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ADEETC

Codificação de Sinais Multimédia

2º Semestre 2016/2017

1º Trabalho prático

João Santos nº 39348

Rui Santos nº 39286

Conteúdo

Lista de Figuras

1 Introdução

O primeiro trabalho prático da disciplina de Codificação de Sinais Multimédia tem como objetivo a utilização da biblioteca OpenCV (Open Source Computer Vision Library) para processamento de imagens através da linguagem Python. Através da utilização desta biblioteca é pretendido realizar operações em imagens, tais como, alterações de qualidade de imagens, conversões de imagens para níveis de cinzento e operações morfológicas com valores de pixeis. O primeiro trabalho prático serve como ambientação há biblioteca OpenCV pois esta biblioteca irá ser utilizada nos próximos trabalhos práticos.

2 Desenvolvimento

Este trabalho prático tem como fio de desenvolvimento sete questões que devem ser respondidas e os seus resultados analisados. Seis das questões presentes no enunciado do trabalho são referentes à imagem de referência "*Lena*", imagem essa que pode ser observada na figura 1.

Figura 1: Imagem original "Lena".

Tendo em conta o referido anteriormente iremos agora delinear cada um dos pontos apresentados no enunciado e as respetivas resoluções.

2.1 Imagem original

De modo a implementar o esperado foi utilizada a biblioteca OpenCV, mais especificamente o método imread("pathname"). O código implementado, tal como, apresentado no enunciado foi o seguinte:

```
lena = cv2.imread("images/lenac.tif")
cv2.imshow("Lena original", lena)
print "dtype: ", lena.dtype
print "shape: ", lena.shape
```

Observando os resultados da execução do código observou-se que a imagem resultado do imshow() é igual à figura 1 e os resultados do "dtype"e "shape"foram os seguintes:

```
dtype: uint8
shape: (512,512,3)
```

O campo dtype = uint8 tem como significado a quantidade de bits utilizada para realizar a codificação da imagem, neste caso 8 bits para cada pixel.
Por sua vez o campo shape = (512,512,3) representa a dimensão da imagem, ou seja, 512x512 e o número 3 representa os planos de cor utilizado, neste caso, RGB.

2.2 Tamanhos dos ficheiros, taxas de compressão, SNR e PSNR

De modo a responder à segunda questão do enunciado foram implementadas as seguintes linhas de código:

```
cv2.imwrite('images/lena - 80%.jpg', lena, (cv2.IMWRITE_JPEG_QUALITY,80))
cv2.imwrite('images/lena - 10%.jpg', lena, (cv2.IMWRITE_JPEG_QUALITY,10))
```

Estas duas linhas de código permitem transformar a imagem original "lenac.tif"para uma imagem com qualidade de 80% e 10% no formato JPEG. De seguida foram efetuados prints dos tamanhos das imagens e observados os resultados.

Tamanho da imagem original: 786814
Tamanho da imagem com 80% de qualidade: 44196
Tamanho da imagem com 10% de qualidade: 9558

Observando os valores obtidos é intuitivo que reduzindo a qualidade da imagem reduzimos também o tamanho do seu ficheiro, isto porque, a quantidade de bits utilizada para codificar a imagem irá ser menor, dai o tamanho do ficheiro ser menor.

É possível observar através das figuras 2 e 3 as imagens com qualidades de 80% e 10%, respetivamente:

Figura 2: Qualidade de 80%.

Figura 3: Qualidade de 10%.

Obtendo ambas as imagens com qualidades diferentes foram calculadas as Taxas de Compressão, SNR e PNSR.

- Taxa de Compressão

A Taxa de Compressão é uma medida de desempenho que é obtida através da razão entre o tamanho da imagem original e o tamanho da imagem após compressão. Uma Taxa de Compressão com valores menores significa que a imagem comprimida tem dimensão aproximada da imagem original, ou seja, não existiu uma compressão acentuada.

$$\text{Taxa de compressão} = \frac{\text{Tamanho da imagem original}}{\text{Tamanho da imagem comprimida}}$$

Calculando as Taxas de Compressão para as qualidades de 80% e 10%, respetivamente, foram obtidos os seguintes resultados:

Taxa de Compressão de uma imagem com 80% de qualidade : 2.44
Taxa de Compressão de uma imagem com 10% de qualidade: 11.27

Tendo em conta o que foi explicado anteriormente podemos observar que como a primeira imagem tem uma qualidade de 80% a sua Taxa de Compressão irá tomar uma valor menor que a imagem com qualidade de 10%.

- SNR (Sinal-to-Noise Ratio)

Seguidamente, foram efetuados os cálculos da SNR(Signal-to-noise ratio). Como o nome indica, a relação sinal-ruído é definida como a razão da potência de um sinal e a potência do ruído sobreposto ao sinal, ou seja, quanto maior o valor da SNR melhor será a relação entre a qualidade da imagem original e da imagem comprimida. O cálculo da SNR é efetuado utilizando a seguinte fórmula:

$$SNR = 10\, log_{10}\, \frac{Psinal}{Pruido}$$

Calculando as SNR's para as qualidades de 80% e 10%, respetivamente, foram obtidos os seguintes resultados:

SNR de uma imagem com 80% de qualidade: 30.9
SNR de uma imagem com 10% de qualidade: 22.87

Como esperado o valor da relação sinal-ruído irá ter uma valor mais elevado para a imagem com uma qualidade mais aproximada da imagem original.

- PSNR (Peak Sinal-to-Noise Ratio)

Finalmente foram efetuados os cálculos da PSNR (Peak Sinal-to-Noise Ratio), a PSNR é um termo utilizado para definir a relação entre a energia máxima de um sinal e o ruído que afeta a sua compressão. A PSNR é mais facilmente definida utilizando o MSE (Mean Squared Error), para tal, foram utilizadas as seguintes expressões matemáticas.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

O MSE é definido utilizando uma imagem monocromática I $m \times n$ sem ruído e uma imagem K com ruído.

$$PSNR = 20\, log_{10}(MAX_I) - 10\, log_{10}(MSE)$$

Observando a expressão da PSNR o valor de MAX_I representa o valor máximo de pixel na imagem, ou seja, quando os pixeis são representados usando 8 bits, o valor máximo é 255. Utilizando ambas as expressões os valores obtidos foram os seguintes:

PSNR com qualidade de imagem de 80%: 31.27
PSNR com qualidade de imagem de 10%: 23.24

O valores típicos de PSNR para imagem com compressão lossy utilizando 8 bits de codificação estão entre 30 e 50 dB, logo, podemos considerar que os nossos valores de PSNR se encontram dentro do esperado.

2.3 Imagem em níveis de cinzento

O terceiro objetivo do trabalho tem como objetivo a conversão da imagem original para uma em níveis de cinzento. A conversão para níveis de cinzento é efetuado utilizando o seguinte método do OpenCV:

lenaGray = cv2.cvtColor(lena , cv2.COLOR_BGR2GRAY)

A utilização deste método implementa a expressão $Y = R \times 299/1000 + G \times 587/1000 + B \times 114/1000$, sendo que, o RGB corresponde a cada componente de cor RED, GREEN e BLUE. A expressão foi obtida devido ao facto da visão humana ser mais sensível ao verde e menos sensível ao azul, adiciona-se então 30% do vermelho mais 59% do verde e apenas 11% do azul. Aplicando esta transformação a imagem obtida pode ser observada na figura 4.

Figura 4: Imagem em tons de cinzento.

Posteriormente à transformação da imagem foi obtido o tamanho do ficheiro da mesma e comparado com o tamanho da imagem original, deste modo, os resultados obtidos foram os seguintes:

Tamanho da imagem original: 786814
Tamanho da imagem em níveis de cinzento: 91229

Observando os valores obtidos e a transformação aplicada é compreensível que o tamanho da imagem em níveis de cinzento irá ser menor que o da imagem original pois são necessários muito menos bits para a codificação da mesma.

2.4 Histograma da imagem em níveis de cinzento

O quarto ponto do trabalho requer a apresentação do histograma da imagem em tons de cinzento, para tal, foi utilizado o seguinte método da biblioteca matplotlib:

plt.hist(lenaGray.ravel(), 256, [0,256], normed = 1, facecolor = 'red')

Implementada a linha de código apresentada em cima o histograma obtido pode ser observado na figura 5.

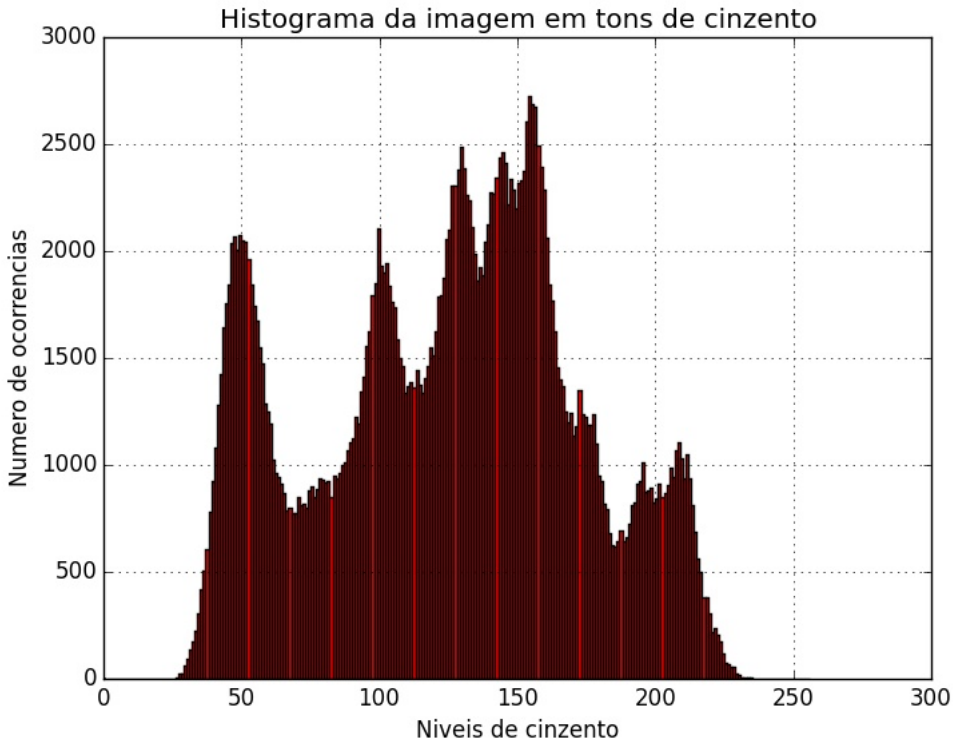

Figura 5: Histograma.

O histograma representa no eixo do x o número de níveis de cinzento e no eiro dos y o número de ocorrências de cada nível. É possível determinar a olho "nu"que não existem pixeis com o valor 0 (preto), nem pixeis com o valor 255 (branco). De maneira a contabilizar o número de níveis de cinzento foi utilizado o método da biblioteca numpy np.unique(lenaGray).size, este método retorna um array com todos os valores existentes sem repetições, ou seja, o comprimento desse array irá ser o número de níveis de cinzento

existentes na imagem. É necessário ter em consideração que o valor 0 não é um nível de cinzento, logo, se existisse uma ocorrência desse valor esta tinha que ser retirada, assim o valor obtido de níveis de cinzento foi 215.

2.5 Manipulação de pixeis na imagem

No quinto ponto do trabalho é necessário realizar manipulação de pixeis. Existem pixeis com maior ou menor peso, ou seja, se considerarmos que o bit de maior peso é o $2^7 = 128$, isto permite que grande parte da informação seja mantida, ao contrario de um bit de menor peso que, como veremos mais à frente, a imagem fica impercetível. A manipulação de pixeis pode ser realiza através de shifts ou intersecções lógicas, deste modo é possível implementar a operação lenaGray & 180 para obter o pixel mais significante da imagem. É possível representar os restantes pixeis menos significantes realizando interseções lógicas usando os bits de menor peso. Um exemplo desta manipulação pode ser o seguinte, o primeiro bit da imagem tem um valor de 160, ou, 10100000 em binário. Realizando uma interceção lógico com o valor 100000000 (128) resulta o valor 128. Deste modo podemos observar que ainda se manteve parte da informação inicial. Caso a operação fosse realizada entre o valor 160 e o valor 16 (00010000) o resultado seria 0 (00000000), logo não ia existir nenhum bit ativo nesse pixels. A figura 6 representa os bits, $2^7, 2^6, 2^5, 2^4, 2^3, 2^2, 2^1$ e 2^0, respetivamente.

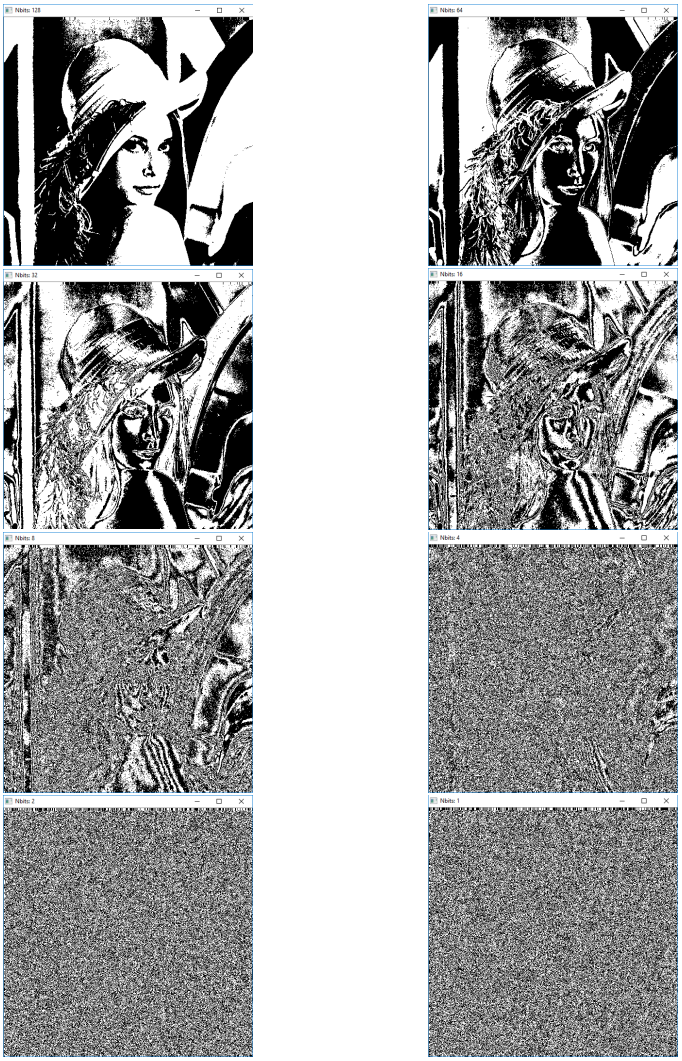

Figura 6: Imagens com a respetiva manipulação de bits.

2.6 Manipulação dos quatro bits mais significantes da imagem

A questão 6 do trabalho prático ainda vem salientar mais o que foi explicado no ponto anterior, ou seja, desta vez realizamos a operação de interseção utilizando os quatro bits mais significantes e observamos que a imagem obtida é visivelmente semelhante à imagem original. Utilizando o mesmo exemplo do ponto anterior mas desta vez com os quatro bits mais significantes obtemos o seguinte $\frac{10100000}{11110000}\& = 10100000 = 160$. O valor resultante da operação foi o mesmo valor inicial do pixel, ou seja, toda a informação foi mantida.

Observado a figura 7 e 8 a comparação da imagem original e da imagem com os quatro bits mais significantes.

Figura 7: Imagem original.

Figura 8: Imagem alterada.

Comparando as duas imagem é possível concluir que utilizando apenas os quatro bits mais significantes a imagem fica percetível e muito aproximada da imagem original.

2.7 Geração da Imagem

Na última alinea do TP, é-nos pedido a criação de uma função que tenha como objectivo, gerar uma imagem idêntica à figura representada no enunciado. Com ajuda da biblioteca *openCV* e as suas funções de desenho:

line(imagem,pontoPartida,PontoChegada,Cor,Espessura)

podemos facilmente deduzir que o nosso *pontoPartida* será o centro da nossa imagem, visto ser uma coordenada estática.

Para o cálculo da coordenada dinâmica *pontoChegada*, recorremos às funções presentes no *numpy* - *cos()* para o cálculo da coordenada X, *sin()* para o cálculo da coordenada Y.

Sendo a expressão de cada coordenada a seguinte:

$$pontoChegadaX = pontoPartidaX + imagemWidth * cos(iteration * angulo * \pi/180)$$

$$pontoChegadaY = pontoPartidaY + imagemWidth * sin(iteration * angulo * \pi/180)$$

É importante referir o uso da variável *iteration*, como queremos que se repita o ângulo escolhido pelo utilizador o número de vezes possíveis em 360, calculamos esse mesmo número de repetições

$$\text{repetições} = 360/angulo$$

e em seguida criamos um loop finito calculando a cada iteração o novo ponto.

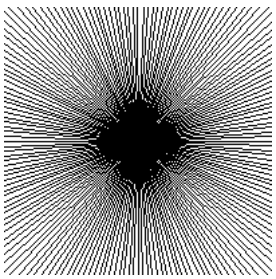

Figura 9: Imagem Gerada - ângulo 2º.

3 Conclusões

Com a finalização do trabalho foram obtidos conhecimentos de métodos da biblioteca OpenCV que vão ser úteis nos trabalhos seguintes. A manipulação de bits de imagens binárias(ou outro tipo de média) é algo que vai ser essencial para a codificação e descodificação de sinais. É necessário saber calcular valores de SNR, taxas de compressão ou outros cálculos que nos ajudem a avaliar o nosso codificar/descodificador. O trabalho serviu como uma introdução aos mecanismos que vão ser utilizado nos trabalhos seguintes. Em suma o trabalho foi realizado com sucesso e os resultados estão dentro dos esperados.