06_Cuantizacion_Laboratorio

April 20, 2021

[1]:
```
import scipy
import scipy.ndimage
from scipy.cluster.vq import vq, kmeans, whiten
from scipy import misc

import numpy as np
import time
import matplotlib.pyplot as plt
import imageio
import PIL
import pickle
```

0.0.1 Lectura y visualización de una imagen

[3]:
```
imagen=imageio.imread('../standard_test_images/jetplane.png')

(n,m)=imagen.shape # filas y columnas de la imagen

plt.figure()
plt.xticks([])
plt.yticks([])
plt.imshow(imagen, cmap=plt.cm.gray,vmin=0, vmax=255)
plt.show()
print(imagen)
```


```
[[193 185 178 … 165 164 147]
 [191 195 194 … 178 156 139]
 [193 197 194 … 190 156 140]
 …
 [215 212 208 … 205 208 212]
 [217 211 208 … 192 188 191]
 [213 211 211 … 184 176 149]]
```

1 Cuantización escalar

Definid una función `Cuantizacion_uniforme_adaptativa(imagen, bits=3, n_bloque=8)` **que dada una imagen cuantize uniformemente los valores en cada bloque.**

```
imagen: imagen a cuantizar
bits: número de bits necesarios para cuantizar cada bloque,
      o sea que en cada bloque habrá 2**bits valores diferentes
      como máximo
n_bloque: se consideran bloques de tamaño n_bloque*n_bloque

imagenCodigo: es una lista de la forma
[[n,m,n_bloque,bits],[[minimo,maximo],bloqueCodificado],...,[[minimo,maximo],bloqueCodificado]]

siendo:
[n,m,n_bloque,bits] información de la imagen
  n: número de filas de la imagen
  m: número de columnas de la imagen
  n_bloque: tamaño de los bloques usados (divisor de n y m)
```

```
  bits: número de bits necesarios para codificar los
        niveles de recontrucción.
Ejemplo: [1024, 1024, 8, 3]

[[minimo,maximo],bloqueCodificado] información de cada bloque codificado
  minimo: valor mínimo del bloque
  maximo: valor máximo del bloque
  bloqueCodificado: array de tamaño n_bloque*n_bloque que contiene
       en cada posición a que intervalo de cuantización correspondía
       el valor del píxel correspondiente en la imagen

Ejemplo: sabemos que trabajamos con bloques 8x8 y que hemos cuantizado en 2**3=8 niveles
    [[85, 150],
    Array([[4, 0, 0, 4, 7, 7, 6, 7],
           [4, 3, 1, 1, 4, 7, 7, 6],
           [6, 6, 3, 0, 0, 4, 6, 6],
           [6, 6, 5, 3, 1, 0, 3, 6],
           [6, 5, 6, 6, 4, 0, 0, 3],
           [5, 6, 6, 6, 6, 4, 2, 0],
           [6, 6, 5, 5, 6, 7, 4, 1],
           [6, 6, 5, 5, 5, 6, 6, 5]]
   El valor mínimo de los píxeles del bloque era 85 y el máximo 150, por lo
   tanto los límites de decisión son:
       [85.0, 93.25, 101.5, 109.75, 118.0, 126.25, 134.5, 142.75, 151.0]
   el valor del primer pixel (4) estaría entre 109.75<=p<118
   el valor del segundo pixel pixel (0) estaría entre 85<=p<93.25...


Importante: Trabajar con Arrays de Numpy
```

[3]:
```
def Cuantizacion_uniforme_adaptativa(imagen, bits=3, n_bloque=8):

    return imagenCodigo
```

Definid una función `Dibuja_imagen_cuantizada(imagenCodigo)` **que dada una imagen codificada por la función** `Cuantizacion_uniforme_adaptativa` **muestre la imagen.**

[3]:
```
def Dibuja_imagen_cuantizada(imagenCodigo):

    return imagen
```

Aplicad vuestras funciones a las imágenes que encontraréis en Atena y haced una estimación de la ratio de compresión.

2 Cuantización vectorial

IMPORTANTE: K-means clustering and vector quantization

http://docs.scipy.org/doc/scipy/reference/cluster.vq.html

Usad las funciones implementadas, en particular vq y kmeans

Definid una funcion `Cuantizacion_vectorial_KMeans(imagen, entradas_diccionario=2**8, n_bloque=8)` **que dada una imagen la cuantize vectorialmente usando K-means.**

```
imagen: imagen a cuantizar
entradas_diccionario: número máximo de entradas del diccionario
       usado para codificar.
n_bloque: Las entradas del diccionario serán bloques de la imagen
       de tamaño n_bloque*n_bloque

imagenCodigo: es una lista de la forma
[[n,m,n_bloque],Diccionario,indices]

siendo:
[n,m,n_bloque] información de la imagen
  n: número de filas de la imagen
  m: número de columnas de la imagen
  n_bloque: tamaño de los bloques usados (divisor de n y m)
Ejemplo: [1024, 1024, 8]

Diccionario: lista de arrays cuyos elementos son bloques de la
     imagen que se usan como diccionario para cuantizar
     vectorialmente la imagen.
Ejemplo:
    [
    array([[173, 172, 172, 171, 171, 171, 171, 172],
       [173, 172, 172, 172, 171, 171, 171, 171],
       [172, 172, 172, 172, 171, 171, 170, 170],
       [172, 172, 171, 171, 171, 171, 170, 169],
       [172, 171, 171, 171, 171, 171, 170, 169],
       [171, 171, 170, 170, 170, 170, 170, 169],
       [171, 171, 170, 170, 170, 170, 169, 169],
       [171, 171, 171, 170, 170, 169, 169, 169]], dtype=uint8),
    array([[132, 131, 128, 122, 118, 117, 121, 124],
       [129, 132, 132, 128, 122, 119, 118, 119],
       [122, 128, 133, 133, 128, 123, 119, 116],
       [115, 121, 128, 131, 132, 130, 124, 119],
       [114, 117, 122, 126, 131, 134, 131, 126],
       [109, 114, 118, 122, 127, 133, 135, 132],
       [ 91, 102, 113, 117, 121, 127, 132, 133],
       [ 70,  89, 107, 114, 115, 120, 127, 131]], dtype=uint8)
...]
```

```
indices: array que contiene los índices de los elementos del diccionario
    por los que hay que sustituir los bloques de la imagen.
Ejemplo: array([14, 124, 22, ...,55, 55, 356], dtype=int32)]
    Al reconstruir la imagen el primer bloque se sustituirá por el bloque 14
    del diccionario, el segundo se sustituirá por el bloque 124 del
    diccionario,..., el último se substituirá por el bloque 356 del
    diccionario.


Importante: Trabajar con Arrays de Numpy
```

[ ]:
```
def Cuantizacion_vectorial_KMeans(imagen, entradas_diccionario=2**8,␣
↪n_bloque=8):

    return imagenCodigo
```

Definid una función `Dibuja_imagen_cuantizada_KMeans(imagenCodigo)` **que dada una imagen codificada por la función** `Cuantizacion_vectorial_KMeans()` **muestre la imagen codificada.**

[ ]:
```
def Dibuja_imagen_cuantizada_KMeans(imagenCodigo):

    return imagen
```

Aplicad vuestras funciones a las imágenes que encontraréis en Atena y haced una estimación de la ratio de compresión.

3 Algunas sugerencias que pueden ser útiles

Divido todos los píxeles de la imagen por q;

a continuación redondeo todos los píxeles;

a continuación sumo $\frac{1}{2}$ a todos los píxeles de la imagen;

a continuación convierto los valores de todos los píxeles en enteros de 8 bits sin signo;

por último múltiplico todos los píxeles de la imagen por q.

[3]:
```
bits=3
q=2**(bits)
imagen2=((np.floor(imagen/q)+1/2).astype(np.uint8))*q
```

dibujo la imagen cuanzizada resultante

[3]:
```
fig=plt.figure()
fig.suptitle('Bloques: '+str(bits)+' bits/píxel')
plt.xticks([])
plt.yticks([])
```

```
plt.imshow(imagen2, cmap=plt.cm.gray,vmin=0, vmax=255)
plt.show()
```

Lectura y escritura de objetos

[3]:
```
import pickle

fichero='QScalar'

with  open(fichero+'_dump.pickle', 'wb') as file:
    pickle.dump(imagenCodigo, file)


with open(fichero, 'rb') as file:
    imagenLeidaCodificada=pickle.load(file)
```

Convertir un array en imagen, mostrarla y guardarla en formato png.

La calidad por defecto con que el ecosistema python (ipython, jupyter,...) muestra las imágenes no hace justicia ni a las imágenes originales ni a las obtenidas tras la cuantización.

[5]:
```
import PIL

imagenPIL=PIL.Image.fromarray(imagen)
imagenPIL.show()
imagenPIL.save('imagen.png', 'PNG')
```

[ ]:
```

```