Práctica 8: Red de Kohonen (Compresión de imágenes) Modelos de la Computación

Emilio Gomez Esteban

1. Enunciado de la prática

Descarga del campus virtual el fichero Compresion.zip que contiene las imágenes Baboon, Lena, Peppers y los archivos correspondientes para realizar la compresión de imágenes.

El script ComprimirImagen.m es el principal, entrena un mapa autoorganizado SOFM de topología cuadrada y tamaño 16×16 con muestras de entrada tridimensionales correspondientes a los valores RGB de los pixeles de una imagen. A continuación se representa cada píxel de la imagen original mediante el prototipo más similar del SOFM. No se ejecuta bien porque falta implementar la función CompeticionSOFM.

1. Crea el script "CompeticionSOFM.m" para poder llevar a cabo el programa de forma adecuada. La función devuelve la ganadora de cada uno de los pixeles, es decir, un vector de índices de neuronas ganadoras de tantos elementos como filas x columnas tiene la imagen (OJO: NO una matriz de ganadoras, nótese el reshape que se hace antes y después de calcular las ganadoras). Nota: En el fichero EntrenarSOFM.m puede que haya algunas líneas que "te inspiren" para implementar la función CompeticionSOFM.m.

   - Pega una captura de la ventana de resultados para cada una de las tres imágenes de prueba (cada ventana tiene 4 imágenes).
   - ¿Qué información necesitamos para pintar la imagen comprimida?
   - ¿Por qué podemos decir que estamos comprimiendo las imágenes?

2. Crea el script CalcularError.m, que lea las tres imágenes originales y las tres imágenes descomprimidas y devuelva como resultado los tres vectores de error cuadrático medio que se producen al realizar la compresión. Recuerda:

$$ECM = \frac{1}{N}\sum_{i=1}^{N}(y_i^{original} - y_i^{comprimido})^2$$

   sería el ECM para una componente de color.

- ¿Qué vectores de error cuadrático medio obtienes en cada caso?
- ¿Qué imagen tiene un vector de error cuadrático de mayor módulo? ¿cuál es su valor?
- ¿Por qué crees que esa imagen se comprime peor?
- Fíjate en el vector ECM de la imagen Peppers, ¿qué componente de color tiene más error? ¿A qué crees que es debido?

Notas:

- En el fichero ComprimirImagen.m hay una línea comentada para guardar la imagen comprimida resultante.
- Usa imread para leer ficheros de imagen.
- Convierte a double antes de operar con la matriz que conforma la imagen.

Sube los scripts CompeticionSOFM.m y CalculaError.m

2. Resolución de la práctica

Ejercicio 1

Aquí tenemos la implementación del script CompeticionSOFM.m:

```
1  function [Ganadoras] = CompeticionSOFM(Model,Muestras)
2  [~,NumMuestras] = size(Muestras); %Muestras traspuesta a la
       forma usual, 3 filas y tantas columnas como pixeles
3  Ganadoras = zeros(NumMuestras,1);
4  NumColsMapa = Model.NumColsMapa;
5  NumFilasMapa = Model.NumFilasMapa;
6  NumNeuronas = NumColsMapa*NumFilasMapa;
7
8  for n=1:NumMuestras
9      MiMuestra = Muestras(:,n);
10     DistsCuadrado = sum((repmat(MiMuestra,1,NumNeuronas)-
           Model.Medias(:,:)).^2,1);
11     [~,NidxGanadora] = min(DistsCuadrado);
12     Ganadoras(n) = NidxGanadora;
13 end
```

- Veamos las capturas de las ventanas de resultados por cada uno de las tres imágenes de prueba:

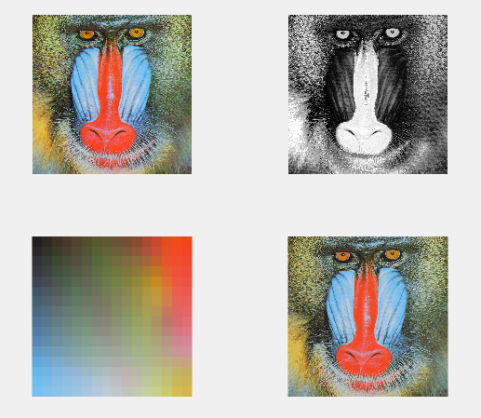

Figura 1: Imagen Baboon

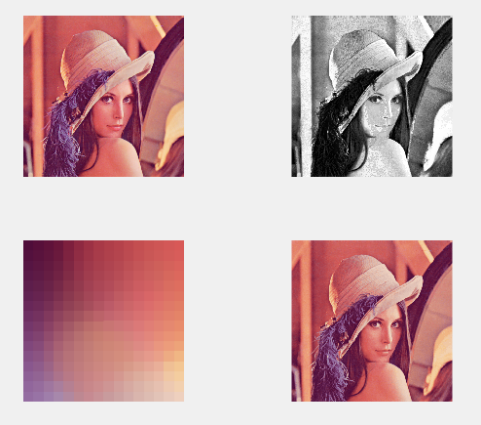

Figura 2: Imagen Lena

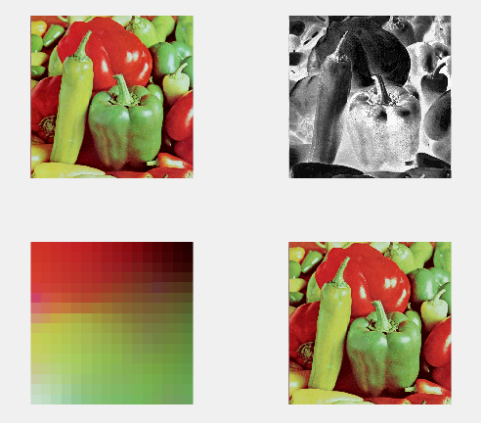

Figura 3: Imagen Peppers

- Para pintar la imagen comprimida, necesitamos: El modelo entrenado del SOFM que contiene los prototipos de cada neurona, la asignación de las neuronas ganadoras para cada píxel (esto se obtiene mediante la competencia del SOFM), los valores de las medias de las neuronas ganadoras, que son las representaciones comprimidas de los colores de los píxeles y la reconstrucción de la imagen comprimida a partir de los valores de las medias, organizados en una matriz tridimensional.

- Estamos comprimiendo la imagen porque reducimos la cantidad de colores posibles que se utilizan para representar la imagen original, mediante la asignación de cada píxel a uno de los prototipos del mapa autoorganizado. Esto permite representar la imagen con menos datos (al usar un número limitado de prototipos en lugar de los valores RGB completos de cada píxel), lo que reduce la redundancia y la dimensionalidad de la información, logrando así la compresión.

Ejercicio 2

Aquí tenemos la implementación del script CalculaError.m:

```
function [MOD,RGB] = CalculaError(Model,Muestras,Ganadoras)
R = norm(Muestras(1,:)-Model.Medias(1,Ganadoras),2);
G = norm(Muestras(2,:)-Model.Medias(2,Ganadoras),2);
B = norm(Muestras(3,:)-Model.Medias(3,Ganadoras),2);
RGB = [R G B];
MOD = sqrt(R*R + G*G + B*B);
end
```

- Vector de ECM[MOD;RGB] para Baboon: [6.1989e+03; 1.0e+03*3.5516, 1.0e+03*3.7603, 1.0e+03*3.4165]
  Vector de ECM[MOD;RGB] para Peppers: [5.3688e+03; 1.0e+03*2.5050, 1.0e+03*2.9146, 1.0e+03*3.7489]
  Vector de ECM[MOD;RGB] para Lena: [3.3584e+03; 1.0e+03*1.8338, 1.0e+03*1.9599, 1.0e+03*2.0187]

- El vector de ECM de Baboon tiene el mayor módulo: 277.34.

- La imagen Baboon se comprime peor porque tiene mayor complejidad visual, menos redundancia espacial, y mayores variaciones en los valores de los píxeles, lo que hace que los algoritmos de compresión no puedan aprovechar tanto las técnicas de reducción de redundancia. Esto da como resultado una imagen comprimida de mayor tamaño y con más pérdida de calidad, lo que se refleja en un mayor módulo de su vector de ECM.

- El componente Azul (B) tiene el mayor error cuadrático medio en el vector ECM para la imagen Peppers, con un valor de 3748.9. Esto se debe a que:

  1. El canal azul probablemente tenga una distribución menos homogénea en la imagen, lo que dificulta la compresión eficiente.

2. Los algoritmos de compresión pueden priorizar los componentes Rojo (R) y Verde (G) debido a la percepción visual humana, dejando más margen de error en el componente Azul (B).
3. El componente azul puede contener más variabilidad y detalles en algunas partes de la imagen, lo que hace que la compresión en ese canal sea más difícil.