IPN
Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Número de la práctica: 3

Arquitectura SOM para generar imágenes en nuevo espacio de color

---

Inteligencia Artificial

Autores:
Alejo Cerezo Braulio Israel
Pérez Monje Juan Pablo
Grupo: 3BM10

Profesor:
Álvaro Anzueto Ríos

Fecha de entrega: 17 de octubre de 2021

1. Introducción

El presente trabajo plantea el desarrollo de un Mapa Autoorganizado (SOM por sus siglas en inglés), aplicado a la extracción de características de color de una imagen, para posteriormente, reconstruir una imagen distinta en este nuevo espacio de color, pues a pesar del gran progreso en el campo de la extración de características, este sigue representando un reto debido a la robustes de los algoritmos empleados. Con lo anterior en mente, las herramientas de Machine Learning permiten desarrollar redes neuronales no supervisadas con la capacidad de conformar, de manera efectiva, complicados mapas de características que superan los espacios tridimensionales, fuera de los cuales, la representación gráfica de los datos es inviable. No obstante, la contraparte de mayor peso de estos algoritmos radica en su elevado costo computacional, pues la cantidad de operaciones necesarias para que la red converja en un resultado adecuado es tal que el tiempo de entrenamiento se vuelve un problema serio en ordenadores de baja y media capacidad.

A pesar de lo ya mencionado, las utilidades de los SOM´s no se limitan únicamente a la extracción de características, sino que también son empleados en algoritmos de cuantización vectorial, reducción de dimensiones y preservación de topología. No obstante, en este documento únicamente se abordará la extracción de tonalidades de imágenes RGB y el posterior ajuste de imágenes totalmente distintas, con el fin de modificar su paleta de colores y obtener una nueva imagen, manteniendo en todo momento la relación de los tonos de los pixeles originales.

Con el fin de visualizar de manera adecuada el papel que desempeñan los SOM´s, se realizaron un total de tres experimentos en los cuales, una misma imagen fue transformada por diferentes SOM, entrenados mediante estilos y tonalidades diversas, uno de ellos siendo una paleta de colores explícita para tonos azules, la cual nos dio un panorama más claro de cómo los mapas autoorganizados pueden realizar *Clustering* y, posteriormente, aplicar esa red neuronal para reconocer y reajustar características de otra fuente de datos. Esto se puede aplicar en algoritmos más robustos con el fin de reconocer estilos artísticos y, generar nuevas obras a partir de los mismos, como se puede observar en los ejemplos realizados cuando se utiliza como entrada a la red una pintura de Vincent van Gogh.

2. Marco Teórico

2.1. Redes competitivas

La redes neuronales competitivas son llamadas de esta manera debido a que cada neurona se excita a si misma e inhibe a todas las demás, analogando esto a una competición por ser el ganador. Este tipo de redes utilizan una función de transferencia específica, la cual actúa como una capa competitiva recurrente, dicha función es:

$$a = \text{compet}(n)$$

Esta función trabaja encontrando el índice i-ésimo de la neurona con la mayor similitud respecto al patrón de entrada, y fija el valor de la salida de esta neurona a 1, a la vez que todas las demás neuronas, llamadas *losers* se fijan en un valor igual a 0.

$$a_i = \begin{cases} 1 & \text{Si } i = i* \\ 0 & \text{Si } i \neq i* \end{cases}$$

No obstante, este tipo de funciones selecciona un conjunto de neuronas de forma *nítida* (valores de 0 o 1), pero también se pueden obtener valores entre 0 y 1 que correspondan a la proporción en que cada neurona es afectada respecto al ganador (valor máximo de 1) y la neurona más alejada del patrón de entrada (valor mínimo de 0)

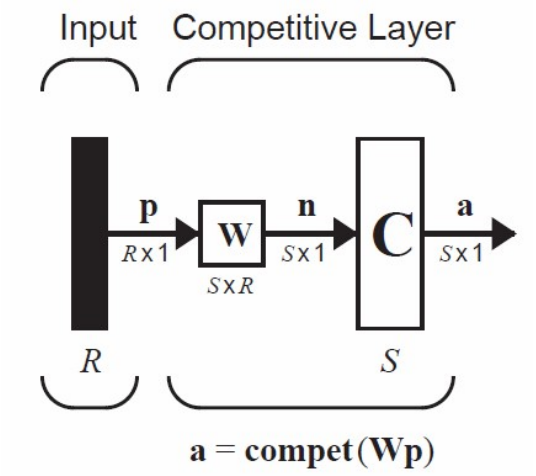


Figura 1: Capa competitiva.

2.1.1. Regla de aprendizaje de Kohonen

Para el diseño de las redes neuronales competitivas se utiliza la regla de aprendizaje de kohonen, pues con ella se tienen los mismos resultados que los obtenidos en arquitecturas tales como INSTAR. De esta manera, tenemos como regla de aprendizaje competitivo:

$$_iw(q) = (1-\alpha)_iw(q-1) + \alpha p(q) \qquad i = i*$$

Y

$$_iw(q) =_i w(q-1) \qquad i \neq i*$$

De esta manera, la matriz de pesos que corresponde a la neurona más cercana al dato de entrada se moverá en dirección del vector de dicho patrón. La distancia que este vector se moverá está en función del factor de aprendzaje α.

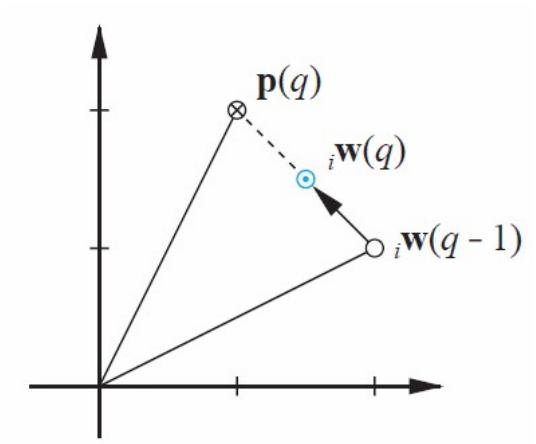


Figura 2: Representación gráfica de la regla de aprendizaje de Kohonen.

2.2. Clustering

El término *clustering* hace referencia a identificar el número de subclases de c agrupamientos (*clusters*) en un universo de datos X comformado por n muestras de información, y separar X en c clusters ($2 \leq c < n$). Cabe mencionar que si $c = 1$ rechaza la hipótesis de que existen clusters en el unverso de información, mientras que $c = n$ constituye el caso trivial en que cada muestra de información se considera un çluster"por sí misma. Existen dos tipos de c-particiones de información: **nítida** o absoluta, y suave o **difusa**.

2.3. Mapas autoorganizados (SOM)

Se ha observado que en el córtex de los animales superiores aparecen zonas donde las neuronas detectoras de rasgos se encuentran topológicamente ordenadas; de forma que la información captada del entorno a través de los órganos sensoriales, se representa internamente en forma de mapas bidimensionales. Aunque normalmente esta organización neuronal está predeterminada genéticamente, es probable que parte de ella se origine mediante el aprendizaje. Esto sugiere, por tanto, que el cerebro podría poseer la capacidad inherente de formar mapas topológicos a partir de la información recibidas del exterior.

También se ha observado que la influencia que una neurona ejerce sobre las demás está en función de la distancia entre ellas, siendo muy pequeña cuando están muy alejadas. El modelo de red auto-organizado presentado por Kohonen pretende mimetizar, de forma simplificada, la capacidad del cerebro de formar mapas topológicos a partir de las señales recibidas del exterior.

Un modelo SOM está compuesto por dos capas de neuronas. La capa de entrada (formada por N neuronas, una por cada variable de entrada) se encarga de recibir y transmitir a la capa de salida, la información procedente del exterior. La capa de salida (formada por M neuronas) es la encargada de procesar la información y formar el mapa de rasgos. Normalmente, las neuronas de la capa de salida se organizan en forma de mapa bidimensional como se muestra en la Figura 3:

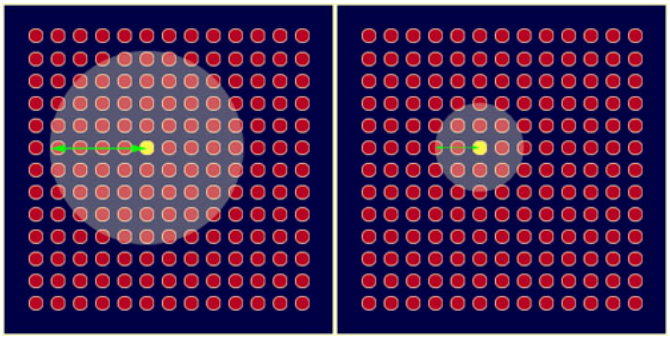


Figura 3: Representación gráfica bidimensional de un mapa de características rectangular, tomando en cuenta el radio del vecindario de una neurona ganadora (amarillo).

2.4. Distancia

Matemáticamente una distancia es una función, $f(a, b)$, que asigna un número positivo a cada par de puntos de un espacio n-dimensional, $a = (a_1, a_2, \ldots, a_n)$, y verifica las siguientes propiedades:

- No negativa, el valor nunca puede ser menor a cero.

$$f(a, b) \geq 0$$

- Simétrica, garantizando que la distancia entre **a** y **b** sea la misma entre **b** y **a**.

$$f(a, b) = f(b, a)$$

- Verifica la desigualdad triangular, las distancia entre dos puntos ha de ser mayor o igual que la suma de las distancias de los puntos originales a un punto intermedio. Es decir, la distancia que hay que recorrer en dos caras de un triángulo es siempre mayor o igual la de la otra cara.

$$f(a, b) \leq f(a, c) + f(c, b)$$

- La distancia con el mismo punto es cero.

$$f(a, a) = 0$$

2.4.1. Distania Euclidiana

La distancia Euclidiana se utiliza para medir la separación entre dos puntos, ya que se interpreta como un espacio de n dimensiones definido mediante la siguiente ecuación:

$$f(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

Aunque esta distancia es útil para medir la separación entre dos puntos en el mundo físico, muestra algunas desventajas cuando se utiliza en un espacio de características, como lo es la dependencia con las unidades de cada una de las coordenadas. A la hora de medir la separación en el mundo físico todas las dimensiones se miden con las mismas unidades (metros, pies, etc.). En un espacio de características generalmente no es así, pues existirán algunas características que tengan más peso (importancia) que otras.

3. Desarrollo

El presente trabajo fue desarrollado a partir de los pasos contemplados en el diagrama de la Figura 4. Es por ello que, a continuación, se expondrán los procesos de generación del código en el lenguage de programación *Python*, el cual corresponde a cada etapa del desarrollo con su respectiva descripción de acuerdo la arquitectura resaltada en el Marco teórico.

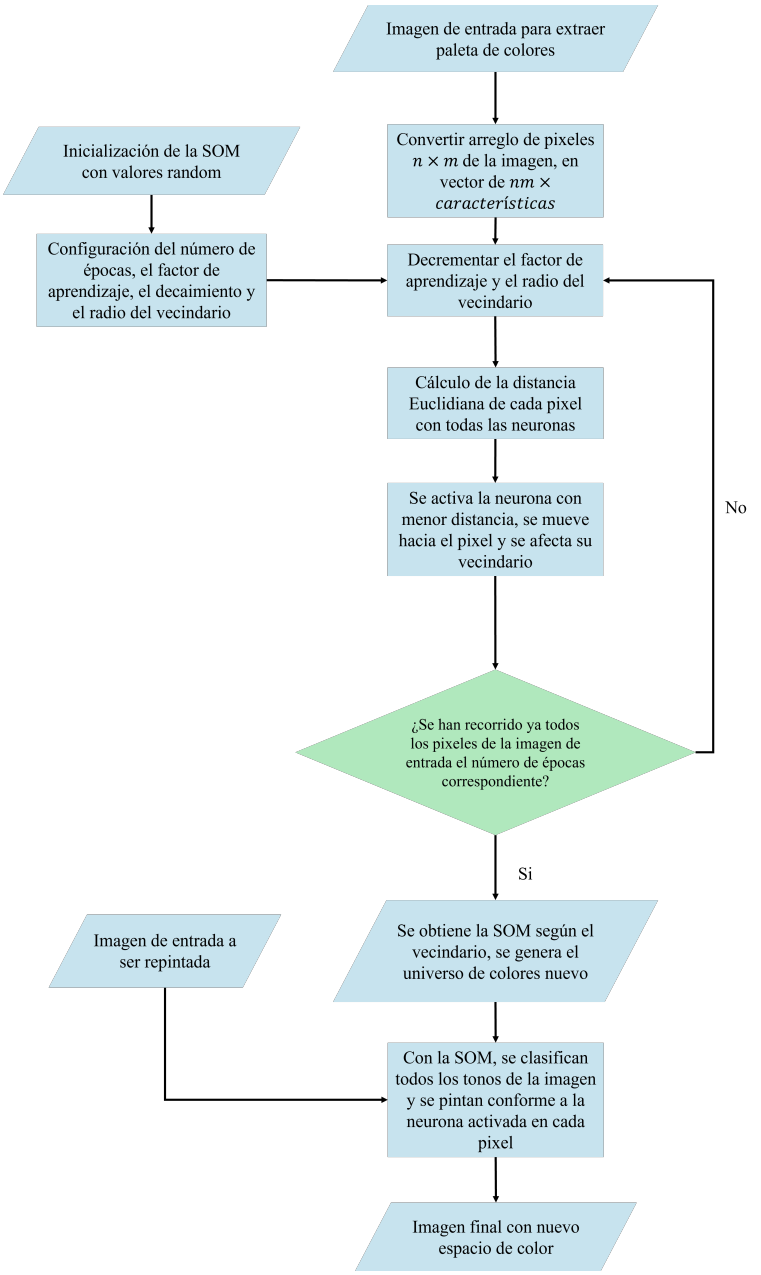


Figura 4: Diagrama de flujo de los subprocesos.

A continuación se describe cada paso del diagrama de subprocesos indicando los respectivos archivos de entrenamiento y el código correspondiente:

Como se observa en el diagrama de flujo, el objetivo de este trabajo es el de diseñar, entrenar y probar la eficacia de un mapa autoorganizado al momento de extraer las características de colores de una imagen. De esta manera, en un primer momento, se debe presentar, como entrada a la red, una imagen que contenga los colores que deseamos extraer mediante el SOM. Para efectos prácticos, se utilizó una paleta de colores en tonalidades azules, como se muestra en la Figura 5.

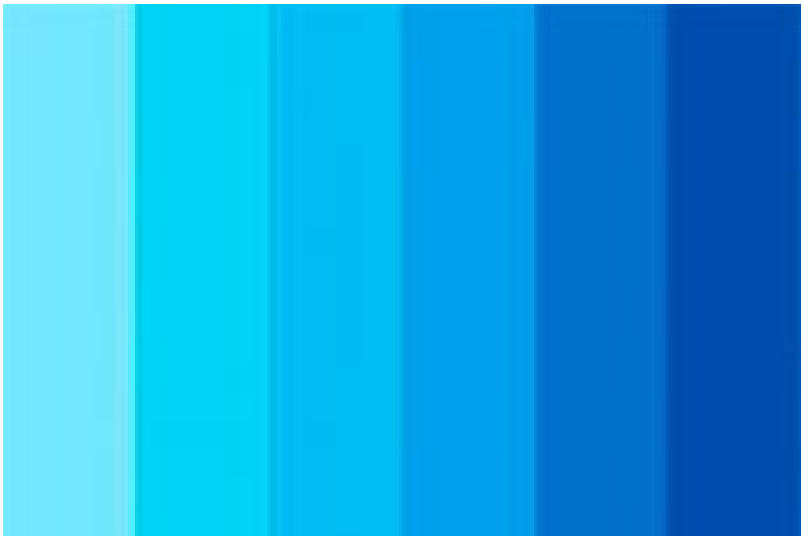


Figura 5: Imagen de entrada al algoritmo del SOM.

3.1. Inicialización del SOM

En primera instancia se cargó la imagen de entrada, nombrada como *paletaAzul.jpg* y la misma se convirtió en un arreglo de $nm \times 3$ datos. Posteriormente se inicializó el mapa de características del SOM con una distribución bidimensional rectangular de 15×15 neuronas, las cuales, de manera ideal se representan tal como lo indica el esquema de la Figura 6. De esta manera se estableció la relación del vecindario de cada neurona, utilizando la distancia Euclidiana existente entre la neurona ganadora y el resto, para obtener $h_{c,i}$. Se debe recordar que el mapa de características representa el total de neuronas de la red, no obstante, cada neurona se compone de un vector de pesos W, el cual se compone a su vez de tres dimensiones, haciendo referencia a los valores de un pixel RGB en la imagen. Por último se inicializaron las variables correspondientes al número de épocas, el factor de aprendizaje, el factor de decaimiento en el tiempo y el radio del vecindario. En la Figura 7 se muestra el mapa autoorganizado con valores aleatorios.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160	161	162	163	164	165
166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195
196	197	198	199	200	201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220	221	222	223	224	225

Figura 6: Representación bidimensional del mapa de características del SOM.

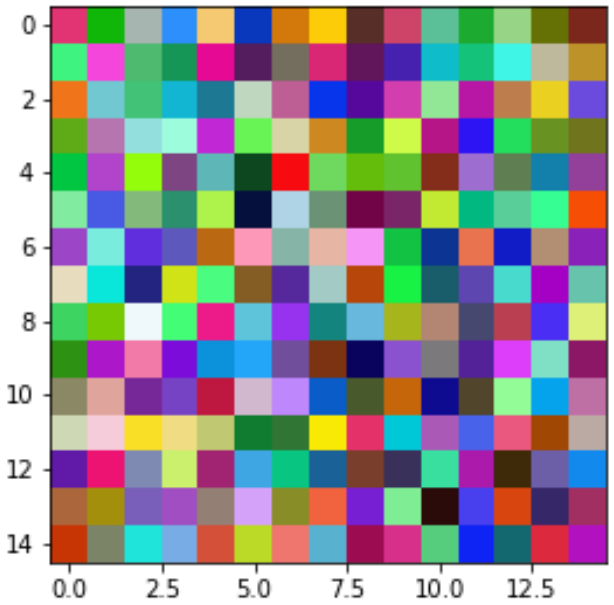


Figura 7: SOM inicializada con valores aleatorios, previo al entrenamiento.

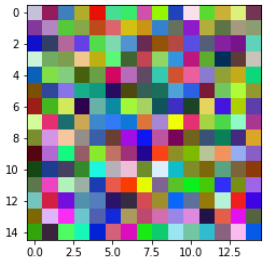
El código código siguiente corresponde a la etapa antes mencionada:

```
nfil = 15
ncol = 15
nras = 3    # Características

datos = io.imread('paletaAzul.jpg')
plt.figure(1)
plt.imshow(datos)

n_total = datos.shape[0]*datos.shape[1]
datos = datos.reshape(n_total, 3) # reordenamiento de la imagen de entrada

# inicializacion del SOM con valores aleatorios
som = np.random.rand(nfil, ncol, nras)
plt.ion()
plt.figure(2)
plt.subplot(1, 2, 1)
plt.imshow(som)

x = np.linspace(0, ncol, ncol)
y = np.linspace(0, nfil, nfil)
x, y = np.meshgrid(x, y)

# inicializacion de variables de la red
epocas = 50
alpha0 = 0.5
decay = 0.05
sgm0 = 20
```

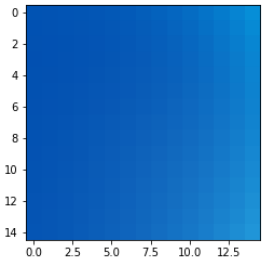
3.2. Entrenamiento del SOM

Una vez cargada la imagen de entrada, y convertida a vector de características RGB, se procedió a generar el código para el entrenamiento del mapa autoorganizado, por lo que se utilizaron dos estructuras for anidadas, con el fin de recorrer el número de épocas de entrenamiento y el número de pixeles correspondientes a la variable *datos*. Dentro de este ciclo anidado se calculó la distancia Eulidiana entre el pixel actual y los pesos sinápticos de cada neurona, estos valores de distancia fueron almacenados en una lista nombrada vec_dists; en cada iteración se compara un único pixel con todas las neuronas, y se encuentra el índice de aquella que presente la mínima distancia en el espacio tridimensional, dicha neurona será considerada como la neurona ganadora, y se moverá en la dirección del pixel actual, todas las neuronas sufrirán un cambio proporcional a su distancia con la neurona ganadora, moviéndose en mayor medida aquellas que se encuentren en el vecindario más próximo a ésta.

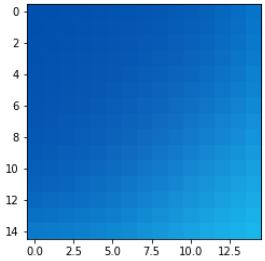
Conforme las neuronas se van ajustando en el mapa autoorganizado, la imagen resultante representa cada vez de mejor manera a las tonalidades identificadas en la imagen de salida, mostrando un proceso de ajuste que se puede observar en la Figura 8 donde se obtuvieron algunos de los resultados más representativos de cada época.



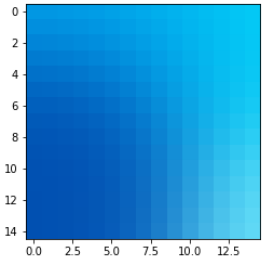
(a) Inicialización.



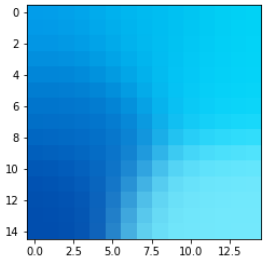
(b) Época 10



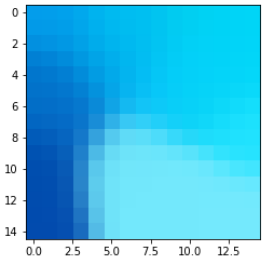
(c) Época 20



(d) Época 30



(e) Época 40



(f) Época 50

Figura 8: Progreso de entrenamiento del mapa autoorganizado.

El código desarrollado para este apartado se presenta a continuación.

```
for t in range(epocas):
    alpha = alpha0 * np.exp(-t * decay)
    sgm = sgm0 * np.exp(-t * decay)
    ven = np.ceil(sgm*3)
```

```
for i in range(n_total):
    vector = datos[i, :]
    columna = som.reshape(nfil*ncol, 3)
    d = 0
    # Cálculo de distancia en RGB
    for n in range(3):
        d = d + (vector[n]-columna[:, n])**2
    vec_dists = np.sqrt(d)

    # Indice para activar la neurona con menor distancia
    ind = np.argmin(vec_dists)
    bmfil, bmcol = np.unravel_index(ind, [nfil, ncol])
    # Inhibir al resto
    g = np.exp( -( ( (x-bmcol)**2) + ((y-bmfil)**2) ) / (2*sgm*sgm) )
    # Radios de acción
    ffil = int( np.max( [0, bmfil-ven] ) )
    tfil = int( np.min( [bmfil+ven, nfil] ) )
    fcol = int( np.max( [0, bmcol-ven] ) )
    tcol = int( np.min( [bmcol+ven, ncol] ) )
    # Modificar SOM
    vecindad = som[ffil:tfil, fcol:tcol, :]
    a, b, c = vecindad.shape
    T = np.ones(vecindad.shape)
    T[:,:,0] = T[:,:,0] * vector[0]
    T[:,:,1] = T[:,:,1] * vector[1]
    T[:,:,2] = T[:,:,2] * vector[2]
    G = np.ones(vecindad.shape)
    G[:,:,0] = g[ffil:tfil, fcol:tcol]
    G[:,:,1] = g[ffil:tfil, fcol:tcol]
    G[:,:,2] = g[ffil:tfil, fcol:tcol]
    # Ecuación de aprendizaje de SOM (regla de aprendizaje de Kohonen)
    vecindad = vecindad + (alpha*G*(T-vecindad))
    # Actualización de som según el vecindario | Genera el universo de colores nuevo
    som[ffil:tfil, fcol:tcol, :] = vecindad
```

3.3. Generación de imagen con el nuevo espacio de color

Para la etapa de prueba del SOM se utilizó una nueva imagen a la cual le fue aplicado el mapa de tonalidades extraído anteriormente. Con ello se diseñó un algortimo que, utilizando los pesos de las neuronas organizadas previamente, pudiera identificar a qué clase pertenece cada pixel de la nueva imagen activando la neurona más cercana y, posteriomente, cambiando la tonalidad de ese pixel por la que corresponde a la neurona ganadora. En esta etapa las reglas de aprendizaje y desplazamiento

de la neurona ganadora y su vecindad fueron eliminadas del código ya que no se requirió reentrenar el mapa de características, sino utilizarlo como sistema de clasificación (clustering) de los datos de la nueva imagen. La imagen utilizada se aprecia en la Figura 9.

Figura 9: Nueva imagen de entrada donde fue aplicado el SOM con le fin de generar un nuevo espacio de color.

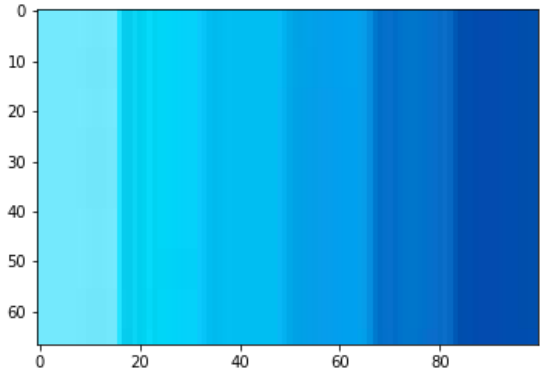
El código para la generación de la nueva imagen con el mapa de color obtenido a partir del archivo *paletaAzul.jpg* se presenta a continuación:

```
#---------------------------------- Testing ------------------------------
datosTest = io.imread('parque.jpg')
paletaColores = som.reshape(nfil*ncol, 3)
newIma = np.zeros((datosTest.shape[0],datosTest.shape[1],3))
for i in range(datosTest.shape[0]):
    for j in range(datosTest.shape[1]):
        pixel = datosTest[i,j,:]
        dist = 0
        # Cálculo de distancia en RGB
        for n in range(3):
            dist = dist + (pixel[n] - paletaColores[:, n])**2
        vectorDists = np.sqrt(dist)
        ind = np.argmin(vectorDists)
        for k in range(3):
            newIma[i,j,k] = paletaColores[ind,k]
```

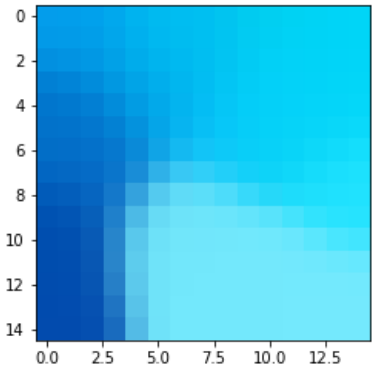
4. Resultados

Una vez ejecutados y completados los algoritmos de mostrados durante el desarrollo, se obtuvieron tres resultados,correspondientes a tres pruebas distintas con diferentes paletas de colores aplicadas a una misma imagen de prueba.

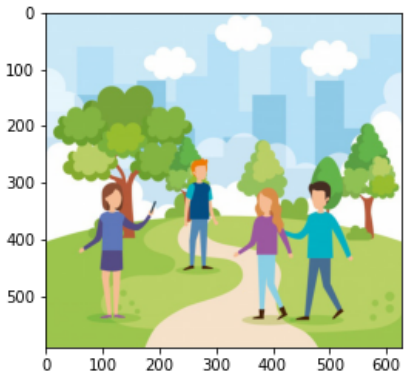
El primer resultado se obtuvo tras haber entrenado el SOM con la imagen *paletaAzul.jpg*, tal como se muestra en el desarrollo. En la Figura 10 se puede apreciar la imágen de entrada para el entrenamiento de la red (10a), el SOM obtenido como resultado del entrenamiento (10b), la imagen de entrada para generar el nuevo espacio de color (10c), y el resultado de aplicar el SOM a dicha imagen (10d).



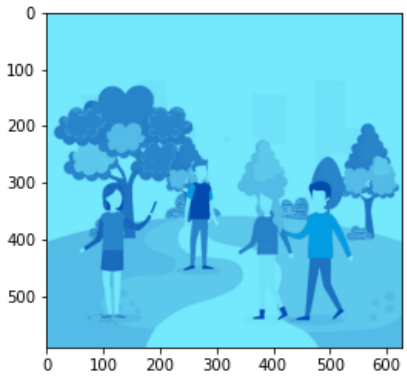
(a) Imagen de entrada.



(b) SOM obtenido.



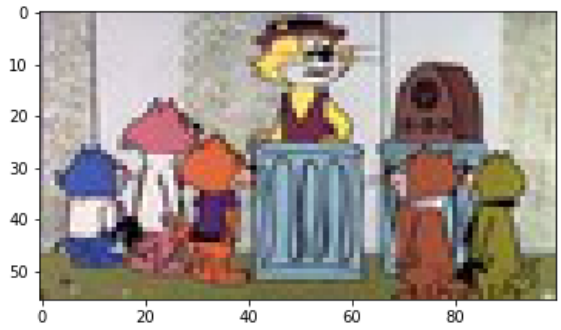
(c) Imagen de prueba.



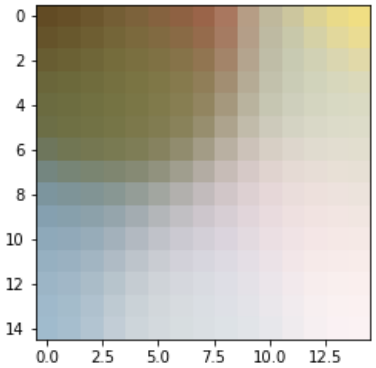
(d) Imagen de salida.

Figura 10: Proceso de obtención de imagen con nuevo espacio de color, prueba número 1.

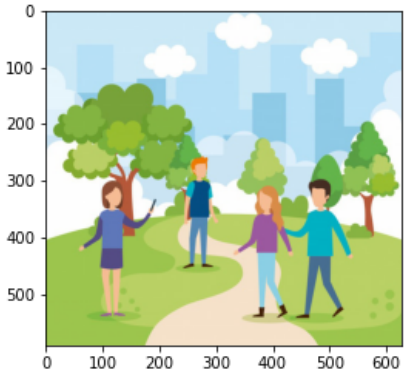
En la segunda prueba el resultado se obtuvo tras haber entrenado el SOM con la imagen *donGato.jpg*. Nuevamente, en la Figura 11 se puede apreciar la imágen de entrada para el entrenamiento de la red (11a), el SOM obtenido como resultado del entrenamiento (11b), la imagen de entrada para generar el nuevo espacio de color (11c), y el resultado de aplicar el SOM a dicha imagen (11d).



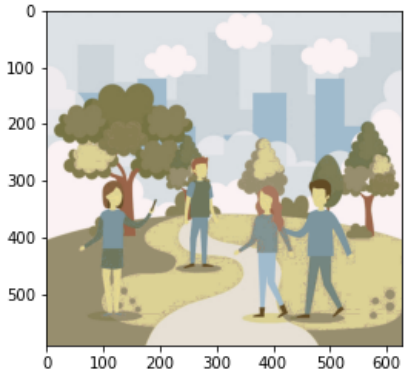
(a) Imagen de entrada.



(b) SOM obtenido.



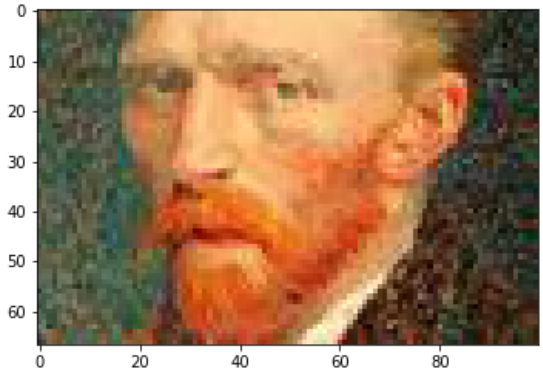
(c) Imagen de prueba.



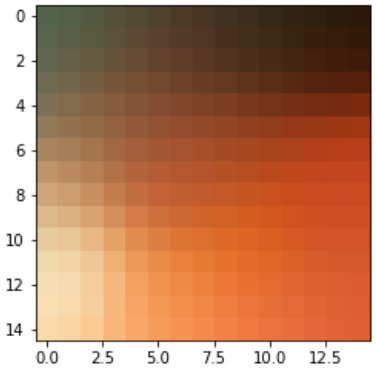
(d) Imagen de salida.

Figura 11: Proceso de obtención de imagen con nuevo espacio de color, prueba número 2.

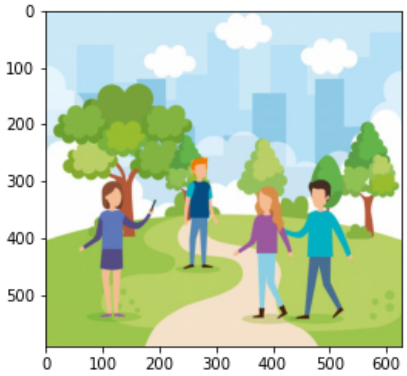
En la tercer prueba el resultado se obtuvo entrenando el SOM con la imagen *VanGogh.jpg*. Una vez más, en la Figura 12 se puede apreciar la imágen de entrada para el entrenamiento de la red (12a), el SOM obtenido como resultado del entrenamiento (12b), la imagen de entrada para generar el nuevo espacio de color (12c), y el resultado de aplicar el SOM a dicha imagen (12d).



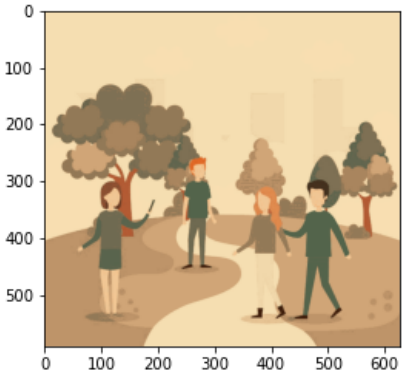
(a) Imagen de entrada.



(b) SOM obtenido.



(c) Imagen de prueba.



(d) Imagen de salida.

Figura 12: Proceso de obtención de imagen con nuevo espacio de color, prueba número 3.

5. Observaciones y/o Conclusiones

Alejo Cerezo Braulio Israel.

El desarrollo de una red neuronal basada en un mapa autoorganizado nos permitió relacionar los conceptos vistos en la práctica de *Clustering* con quellos relacionados con las redes neuronales, pues más allá de los algoritmos de K-Meas y Fuzzy C-Means, los mapas autoorganizados permiten extraer características con algoritmos de aprendizaje no supervisado utilizando un método similar basado en el cálculo de la menor distancia euclidiana entre el dato y todos los centros de clase, que en este caso, fueron representados mediante el vector de pesos W de cada neurona.

El mapa autoorganizado desarrollado en el presente trabajo no solo permitió aplicar agrupamiento de datos a una imagen determinada, sino que los datos clasificados por las neuronas se utilizaron posteriormente para redefinir los colores de una imagen distinta, reorganizando sus pixeles RGB y obteniendo una nuevo conjunto de tonos a la salida. Al trabajar con un algoritmo de esta naturaleza, considerando que las características de cada patrón de entrenamiento se encontraban en un espacio tridimensional, fue posible observar cómo las neuronas se ajustaban a cada pixel y éstas a su vez afectaban a su vecindad más próxima, los cambios por supuesto fueron realizados considerando que todo el mapa de características (mapa de neuronas bidimensional rectangular) se vaía afectado por la neurona ganadora, sin embargo, debido a la distancia de separanción de algunas neuronas con la neurona ganadora, su influencia era prácticamente imperceptible, aunado al hecho de que en cada iteración, el factor de aprendizaje α y el radio de vecindario σ decrementaban a razón del factor de decaimiento.

Pérez Monje Juan Pablo.

A lo largo de esta práctica se presentó el proceso de aprendizaje y clasificación de una red neuronal correspondiente a un mapa autoorganizado de Kohonen. Este algoritmo no sólo se encarga de extraer las características de color de una imagen determinada, sino que también utiliza un rango dinámico de vecindario para ubicar a las neuronas en una posición óptima que represente adecuadamente cada agrupamiento de clase. No obstante, el radio dinámico no fue el único que se reducía conforme al número de entrenamientos, utilizar una tasa de aprendizaje dinámico también permitió que el algoritmo pudiera extraer características con una menor redundancia de datos realizando un ajuste más preciso en los colores predominantes de cada imagen.

Los resultados de la etapa de experimentación nos permitieron comprobar la efectividad de aplicar un mapa autoorganizado para generar un nuevo espacio de colores en una imagen totalmente distinta, obteniendo un mismo cuadro, pintado con tres paletas de colores características. A pesar de que los resultados obtenidos fueron acordes a los esperados en esta práctica, una clara desventaja de este tipo de algoritmos es el tiempo que tardan en converger a un resultado, considerando que, dependiendo de la imagen estudiada, el número de épocas usadas en este trabajo podría no ser suficiente para conseguir resultados adecuados en la distribución de colores (recordando además que las imágenes utilizadas en el entrenamiento de la red SOM fueron reducidas a un tamaño de 100×50 pixeles).

Referencias

Hagan, M., Demuth, H., Beale, M. & De Jesús, O. (2014). *Neural Network Design* .(2nd Edition). 7:1-5, 15:1-17.

Kohonen T. (2001). *Self-organizing maps*. 3rd edn. Springer Ser Inf Sci 30:501.

Ong S., Yeo C., Lee K., Venkatesh V., Cao D. (2002). *Segmentation of color images using a two-stage self-organizing network*. Image Vis Comput 20(4):279–289.

Rasti J., Monadjemi A., Vafaei A. (2011). *Color reduction using a multi-stage Kohonen self-organizing map with redundant features*. Expert Syst Appl 38(10):13188–13197.

Yin H. (2008). *The self-organizing maps: background, theories, extensions and applications*. In: Computational intelligence: a compendium, vol 115. Springer, Heidelberg, pp. 715–762.

6. Anexos

6.1. Código del programa: SOM para generar imágenes en nuevo espacio de color

```
from skimage import io, data, color
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt

nfil = 15
ncol = 15
nras = 3    # Características

# datos = io.imread('paletaAzul.jpg')
# datos = io.imread('donGato1.jpg')
datos = io.imread('vanGogh2.jpg')
plt.figure(1)
plt.imshow(datos)

n_total = datos.shape[0]*datos.shape[1]
datos = datos.reshape(n_total, 3) # input

som = np.random.rand(nfil, ncol, nras)
plt.ion()
plt.figure(2)
plt.subplot(1, 2, 1)
plt.imshow(som)

x = np.linspace(0, ncol, ncol)
y = np.linspace(0, nfil, nfil)
x, y = np.meshgrid(x, y)
epocas = 50
alpha0 = 0.5
decay = 0.05
sgm0 = 20

for t in range(epocas):
    alpha = alpha0 * np.exp(-t * decay)
    sgm = sgm0 * np.exp(-t * decay)
    ven = np.ceil(sgm*3)
```

```
    for i in range(n_total):
        vector = datos[i, :]
        columna = som.reshape(nfil*ncol, 3)
        d = 0
        # Cálculo de distancia en RGB
        for n in range(3):
            d = d + (vector[n]-columna[:, n])**2
        vec_dists = np.sqrt(d)

        # Indice para activar la neurona con menor distancia
        ind = np.argmin(vec_dists)
        bmfil, bmcol = np.unravel_index(ind, [nfil, ncol])
        # Inhibir al resto
        g = np.exp( -( ( (x-bmcol)**2) + ((y-bmfil)**2) ) / (2*sgm*sgm) )
        # Radios de acción
        ffil = int( np.max( [0, bmfil-ven] ) )
        tfil = int( np.min( [bmfil+ven, nfil] ) )
        fcol = int( np.max( [0, bmcol-ven] ) )
        tcol = int( np.min( [bmcol+ven, ncol] ) )
        # Modificar SOM
        vecindad = som[ffil:tfil, fcol:tcol, :]
        a, b, c = vecindad.shape
        T = np.ones(vecindad.shape)
        T[:,:,0] = T[:,:,0] * vector[0]
        T[:,:,1] = T[:,:,1] * vector[1]
        T[:,:,2] = T[:,:,2] * vector[2]

        G = np.ones(vecindad.shape)
        G[:,:,0] = g[ffil:tfil, fcol:tcol]
        G[:,:,1] = g[ffil:tfil, fcol:tcol]
        G[:,:,2] = g[ffil:tfil, fcol:tcol]

        # Ecuación de aprendizaje de SOM (regla de aprendizaje de Kohonen)
        vecindad = vecindad + (alpha*G*(T-vecindad))

        # Actualización de som según el vecindario |
        # Genera el universo de colores nuevo
        som[ffil:tfil, fcol:tcol, :] = vecindad

plt.subplot(1, 2, 2)
plt.imshow(np.uint8(som))
plt.show()
```

```
#----------------------------------- Testing --------------------------------
datosTest = io.imread('parque.jpg')
paletaColores = som.reshape(nfil*ncol, 3)
newIma = np.zeros((datosTest.shape[0],datosTest.shape[1],3))
for i in range(datosTest.shape[0]):
    for j in range(datosTest.shape[1]):
        pixel = datosTest[i,j,:]
        dist = 0
        # Cálculo de distancia en RGB
        for n in range(3):
            dist = dist + (pixel[n] - paletaColores[:, n])**2
        vectorDists = np.sqrt(dist)
        ind = np.argmin(vectorDists)
        for k in range(3):
            newIma[i,j,k] = paletaColores[ind,k]

plt.figure(4)
plt.subplot(1, 2, 1)
plt.imshow(np.uint8(datosTest))
plt.subplot(1, 2, 2)
plt.imshow(np.uint8(newIma))
```