Práctica 3: Caracterización y Clasificación de Texturas

Aguilar Luna Gabriel Daniel, Rodríguez Agiss Zuriel Uzai

***Abstract*—Practica de la clase de Reconocimiento de Patrones que representa el aprendizaje aprendido en cuánto a Caracterización y Clasificación de Texturas**

***Index Terms*—Texturas, K-NN, K-means, máquinas de soporte vectorial.**

I. Objetivo

Desarrollar métodos de caracterización de texturas y aprender a utilizar clasificadores como K-NN,K-Means o máquinas de soporte vectoria

II. Introduction

A. Análisis y caracterización de las Texturas

El análisis de texturas hace referencia a la caracterización de las regiones de una imagen por su contenido de textura. El análisis de texturas intenta cuantificar las cualidades intuitivas descritas por términos como áspero, suave, sedoso o accidentado en función de la variación espacial en las intensidades de píxeles. En este sentido, la rugosidad o bache se refiere a variaciones en los valores de intensidad, o niveles de gris.

El análisis de texturas se utiliza en varias aplicaciones, incluyendo la teledetección, la inspección automatizada y el procesamiento de imágenes médicas. El análisis de texturas se puede utilizar para encontrar los límites de textura, denominados segmentación de texturas. El análisis de texturas puede ser útil cuando los objetos de una imagen se caracterizan más por su textura que por la intensidad, y las técnicas de umbral tradicionales no se pueden utilizar de forma eficaz.

Es posible usar métodos estadísticos para el análisis de texturas como por ejemplo:

- Analizar la distribución espacial de valores de gris es una calidad que define la textura.
- Analizar la distribución espacial de los valores de gris, se computan características locales de la textura. Ejemplos:
  - Media y varianza.
  - Co-ocurrencia y diferencias de niveles de gris.

B. Clasificadores

1) K-NN: El método de los vecinos más cercanos (k-NN) es un algoritmo de aprendizaje automático simple y conocido que utiliza un grupo de datos de entrenamiento ”más cercano” alrededor de un punto de datos de prueba para clasificarlo. En las implementaciones de k-NN más simples, un punto de prueba se clasifica simplemente asignándolo a la clase más común entre los k puntos de entrenamiento más cercanos.

2) K-Means: El algoritmo Kmeans es un algoritmo iterativo que intenta dividir el conjunto de datos en subgrupos (clústeres) distintos no superpuestos definidos previamente por K donde cada punto de datos pertenece a un solo grupo. Intenta hacer que los puntos de datos intra-clúster sean lo más similares posible y al mismo tiempo mantiene los clústeres lo más diferentes (lejos) posible. Asigna puntos de datos a un grupo de modo que la suma de la distancia al cuadrado entre los puntos de datos y el centroide del grupo (media aritmética de todos los puntos de datos que pertenecen a ese grupo) es mínima. Mientras menos variación tengamos dentro de los conglomerados, más homogéneos (similares) serán los puntos de datos dentro del mismo conglomerado.

3) Máquinas de Soporte Vectorial (SVM): Las máquinas de vectores de soporte (SVM) son un conjunto de métodos de aprendizaje supervisado que se utilizan para la clasificación, la regresión y la detección de valores atípicos.

Las ventajas de las máquinas de vectores de soporte son:

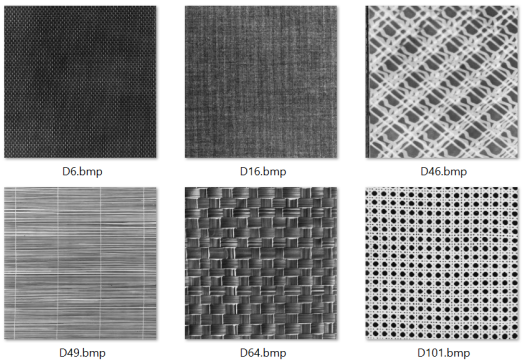
- Eficaz en espacios de gran dimensión.
- Sigue siendo eficaz en los casos en que el número de dimensiones es mayor que el número de muestras.
- Utiliza un subconjunto de puntos de entrenamiento en la función de decisión (llamados vectores de soporte), por lo que también es eficiente en la memoria.
- Versátil: se pueden especificar diferentes funciones del núcleo para la función de decisión. Se proporcionan núcleos comunes, pero también es posible especificar núcleos personalizados.

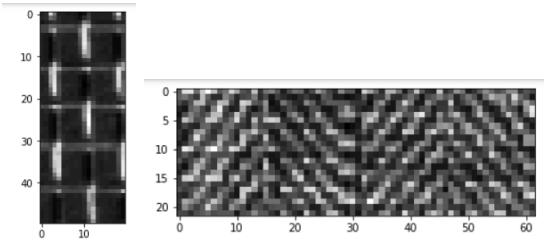
Las desventajas de las máquinas de vectores de soporte incluyen:

- Si el número de características es mucho mayor que el número de muestras, evite el ajuste excesivo al elegir las funciones del Kernel y el término de regularización es crucial.
- Las SVM no proporcionan directamente estimaciones de probabilidad, estas se calculan mediante una costosa validación cruzada de cinco veces (consulte Puntuaciones y probabilidades, a continuación).

III. Desarrollo

1) A partir de un conjunto de texturas que se le proporciona al alumno generar un sistema de “image retrieval” mediante un proceso de reconocimiento de patrones
   Para esta parte se cuentan con las siguientes imágenes
   Se procede a dividir en ventanas por medio de una función llamada WindowMaker. A continuación se muestran ejemplos del output de dicha función Lo que





prosigue ahora es obtener información característica de las imágenes a partir del proceso de extracción de características que entrega la matriz de GLCM. GLCM es una tabulación de la frecuencia con la que ocurren diferentes combinaciones de valores de brillo de píxeles (niveles de gris) en una imagen. A continuación se muestra un ejemplo de su ejecución, recordando que el código de la función se halla en el anexo de código. En

```
In [14]:
imagen_prueba = np.array([[0,0,1,1],[0,0,1,1],[0,2,2,2],[2,2,3,3]])
glcmMaker_h(imagen_prueba)

Out[14]: array([[4, 2, 1, 0],
                [2, 4, 0, 0],
                [1, 0, 6, 1],
                [0, 0, 1, 2]])

In [15]:
glcmH_N = glcmMaker_h(imagen_prueba, True)
glcmH_N

Out[15]: array([[0.16666667, 0.08333333, 0.04166667, 0.        ],
                [0.08333333, 0.16666667, 0.        , 0.        ],
                [0.04166667, 0.        , 0.25      , 0.04166667],
                [0.        , 0.        , 0.04166667, 0.08333333]])
```

el primer ejemplo se retorna la matriz GLCM de una matriz de prueba, y en el segundo se retorna la misma matriz pero normalizada

De esa matriz podemos obtener información que nos permita generar un vector de características. Estos valores son los siguientes:

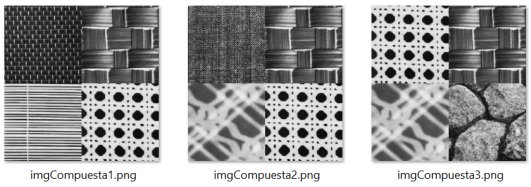
$$Entropia = \sum_{i,j=0}^{N-1} -ln(P_{ij})P_{ij}$$

$$Homogeneidad = \sum_{i,j=0}^{N-1} \frac{P_{ij}}{1+(i-j)^2}$$

$$Energía = \sum_{i,j=0}^{N-1} (P_{ij})^2$$

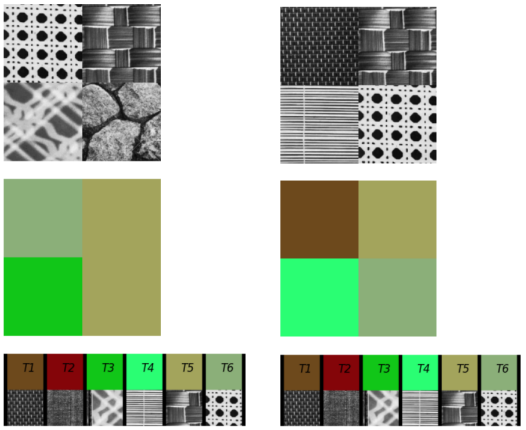
$$Contraste = \sum_{i,j=0}^{N-1} P_{ij}(i-j)^2$$

Una vez se tiene el vector característico es posible comenzar con el proceso de clasificación. Para ello se tienen las siguientes imágenes de prueba

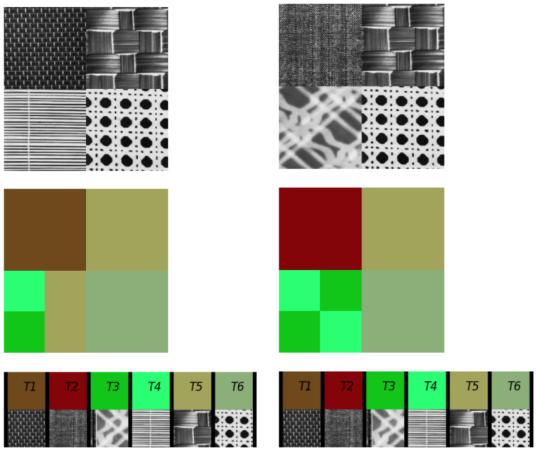


IV. Resultado

Para la práctica se hizo uso de un clasificador bayesiano y uno de K-NN. Con el clasificador bayesiano se obtuvieron los siguientes resultados



Con el clasificador K-NN se obtuvieron los siguientes resultados



V. Conclusiones

Esta práctica nos permitió en primera instancia a usar métdos estadísticos para la caracterización de texturas, y nos permitió darnos cuenta de que una vez entendiendo bien la teoría, la implementación no tenía porque ser tan tediosa puesto que al final del día la computadora era la que estaba

haciendo toda la magía computacional a gran escala. En cuanto a los resultados, nos dimos cuenta de que debemos ser cuidadosos con el tamaño de nuestra muestra y contemplar todo tipo de texturas posibles a la hora de clasificar. También conocimos un método de caracterización llamado K-NN además del bayesiano que ya conocíamos. Por ende, consideramos que los objetivos de la práctica fueron cumplidos.

VI. Código

```
import matplotlib.pyplot as plt
import cv2
import skimage as ski
import PIL as pil
import imageio as io
import numpy as np
from PIL import Image, ImageFilter
from shapely.geometry import Polygon, MultiPolygon
from descartes import PolygonPatch
from math import log
from sklearn.neighbors import KNeighborsClassifier

# Esta funcion 'desdobla' cadenas para crear un rango y un numero
# eg: '100'=mayores que 100, '0-50'= entre 0 y 50, '10-20,500'= entre 10 y 20 o mayores de 500
def listRanger(rango: str):
    # Separa la cadena por comas
    rango = rango.split(',')
    # Definicion de las variables de retorno
    listarango = []
    mayorq = -1
    # Recorre las sentencias separadas anteriormente
    for x in rango:
        # Si la sentencia tiene un '-' es un rango
        if '-' in x:
            # Los rangos se a aden a la lista listarango
            listax = x.split('-')
            listarango += list(range(int(listax[0]), int(listax[1])))
        # Si no, es una cota inferior
        else:
            # Solo puede existir una de estas cotas en la sentencia
            mayorq = int(x)

    return [listarango, mayorq]

# Esta funci n imprime la imagen junto con las curvas de la posici n nColl de contornos
# Que cumplan con range y retorna un arreglo con dichas curvas.
# Se puede especificar los puntos en el perimetro de las lineas con range. eg: range='400-560'
# Si no se especifica se utiliza '150'
# Se puede especificar un nombre para guardar la imagen resultante con save. eg: save='imagen2.png'
# Si no se especifica se utiliza 'ImCrTMP.png'
# eg: masker.printImCr(fruta, fruta_contornos, 1, range='200', save='comida_contornos.png')
def printImCr(imagen, contornos, nColl, **kwargs):
    # Desdobla range
    rangolstd = listRanger(kwargs["range"] if ("range" in kwargs) else '150')
    # Define variable de retorno
    curvas_array = []
    # Con este for se muestran todas las lineas cuya primera dimension entre en el rango
    for i in contornos.collections[nColl].get_paths():
        # Si cumple con las caracteristicas indicadas en range se a ade al plot y a la var de retorno
        if len(i.vertices) in rangolstd[0] or ((len(i.vertices) > rangolstd[1]) if (rangolstd[1] != -1) else (len(i.vertices) > len(i.vertices)+1)):
            plt.plot(i.vertices[:,0], i.vertices[:,1], '--b')
            curvas_array.append(i.vertices)
    # Muestra la imagen
    plt.imshow(imagen)
    plt.axis('off')
    # Salva la imagen
```

```
    plt.savefig('resultados/'+kwargs["save"] if ("save" in kwargs) else 'resultados/ImCrTMP.png', bbox_inches='tight', transparent=False, pad_inches = 0)
    # Regresa el arreglo
    return curvas_array

# Esta funci n imprime la imagen de fondo junto con los poligonos
# Definidos por las curvas en curvas_arr
# Retorna un obj Multipolygon
def printImPoly(curvas_arr, fondo):
    # Arreglo aux
    poly_array = []
    # Recorre arreglo de curvas
    for crvua in curvas_arr:
        x = crvua[:,0]
        y = crvua[:,1]
        poly_array.append(Polygon([(i[0], i[1]) for i in zip(x,y)]))
    polygons = MultiPolygon(poly_array)
    #len(polygons.geoms)
    #polygons
    fig = plt.figure()
    ax = fig.gca()

    # Plotea la imagen de fondo
    plt.imshow(fondo)
    # Plotea los poligonos
    for poly in polygons:
        ax.add_patch(PolygonPatch(poly))
    #ax.axis('scaled')
    plt.show()
    # Retorna los poligonos
    return polygons

def listaclase(clase, n=4):
    clase_array = []
    for i in range(n):
        objeto = io.imread('entrenamiento-procesado/Entrenamiento'+str(i+1)+'-'+ clase +'_gaussianBlur.jpg')
        clase_array.append(np.array(objeto))
        plt.imshow(objeto)
        plt.show()
    return clase_array

def pixelsMatrix(clase, n=4):
    lista_clase = listaclase(clase, n)
    pixeles_clase = []
    for objeto in lista_clase:
        for renglon in objeto:
            for pixel in renglon:
                if pixel.mean() > 0.:
                    pixeles_clase.append(pixel)
    return np.array(pixeles_clase)

def pixelsMatrix_blanco(clase, n=6):
    lista_clase = listaclase(clase, n)
    pixeles_clase = []
    pixeles_probados = 0
    for objeto in lista_clase:
        for renglon in objeto:
            for pixel in renglon:
                pixeles_probados += 1
                if pixel.mean() < 250.:
                    pixeles_clase.append(pixel)
    return np.array(pixeles_clase), pixeles_probados

def getSigma(matriz_pix):
    MU = np.mean(matriz_pix, axis=0)
    SIGMA = np.zeros((len(MU),len(MU)))

    for pixel in matriz_pix:
        P_MU = np.array([pixel-MU])
        SIGMA += np.dot(np.transpose(P_MU),P_MU)

    return SIGMA / len(matriz_pix)

def probaClase(lenClase, x, y, n):
    return lenClase/(x*y*n)

def ladoDerecho(SIGMA_det, probaClase):
    return (-log(SIGMA_det)/2) + log(probaClase)

def probaPixClase(pixel, MU, SIGMA_inv):
    P_MU = np.array([pixel-MU])
    #print((np.dot(np.dot(P_MU,SIGMA_inv), np.transpose(P_MU))[0,0])/-2)
    return (np.dot(np.dot(P_MU,SIGMA_inv), np.transpose(P_MU))[0,0])/-2

def clasificador(imagen, MUs, SIGMAs_inv, ldDers, colores, **kwargs):
    show = (kwargs["show"] if ("show" in kwargs) else False)
    if (isinstance(imagen, str)):
        prueba_img = io.imread(imagen)
        prueba = np.array(prueba_img)
    elif(isinstance(imagen, np.ndarray)):
        prueba = imagen
    if show:
        plt.imshow(prueba)
        plt.show()

    dimensiones = prueba.shape
    resultado = np.zeros((dimensiones[0],dimensiones[1],3), dtype=int)
    for i in range(len(prueba)):
        for j in range(len(prueba[i])):
            arr_aux = []
            for n in range(len(MUs)):
                arr_aux.append(probaPixClase(prueba[i][j], MUs[n], SIGMAs_inv[n])+ldDers[n])
            resultado[i][j]= colores[arr_aux.index(max(arr_aux))]
    return resultado

def makeGaussian(nombre_imagen, valor_radio = 3):
    #codigo de internet para aplicar filtro Gaussiano
    #Abriendo imagen con Image
    image = Image.open(nombre_imagen)

    #Aplicando el filtro de Gaussiano
    #Con un valor para el radio igual a 3
    image = image.filter(ImageFilter.GaussianBlur(radius = valor_radio))

    #Se muestra la imagen
    image.show()

    #Se gurada esta nueva imagen borrosa
    #image.save(nombre_imagen[:-4]+"_gaussianBlur.jpg")
def listaTexturas(text_names):
    texturas_array = []
    for text_name in text_names:
        objeto = io.imread('texturas/D'+text_name+'.bmp')
        texturas_array.append(np.array(objeto)[:,:,0])
        plt.imshow(objeto)
        plt.show()
    return texturas_array
def windowMaker(imagen,window_length,window_height,**kwargs):
    Y, X = imagen.shape
    windows_array = []
    length_overlap = (kwargs["length_overlap"] if ("length_overlap" in kwargs) else 0)
    height_overlap = (kwargs["height_overlap"] if ("height_overlap" in kwargs) else 0)
```

```
    show = (kwargs["show"] if ("show" in kwargs) else False)
    for texel_renglon in range(0,Y,window_height-height_overlap):
        for texel_startP in range(0,X,window_length-length_overlap):
            ventana = imagen[texel_renglon:texel_renglon+window_height,texel_startP:texel_startP+window_length]
            windows_array.append(ventana)
            if (show):
                plt.imshow(ventana, cmap=plt.cm.gray)
                plt.show()
    return windows_array


def glcmMaker_v(ventana, norm = False):
    matriz_auxiliar = np.zeros((np.amax(ventana)+1,)*2, dtype=int)
    Y, X = ventana.shape
    normalizador = 0
    for i in range(Y-1):
        for j in range(X):
            matriz_auxiliar[ventana[i][j]][ventana[i+1][j]] += 1
            normalizador += 2
    glcm = matriz_auxiliar + np.transpose(matriz_auxiliar)
    return (glcm/normalizador) if norm else glcm

def glcmMaker_45(ventana, norm = False):
    matriz_auxiliar = np.zeros((np.amax(ventana)+1,)*2, dtype=int)
    Y, X = ventana.shape
    normalizador = 0
    for i in range(1,Y):
        for j in range(X-1):
            matriz_auxiliar[ventana[i][j]][ventana[i-1][j+1]] += 1
            normalizador += 2
    glcm = matriz_auxiliar + np.transpose(matriz_auxiliar)
    return (glcm/normalizador) if norm else glcm

def glcmMaker_135(ventana, norm = False):
    matriz_auxiliar = np.zeros((np.amax(ventana)+1,)*2, dtype=int)
    Y, X = ventana.shape
    normalizador = 0
    for i in range(1,Y):
        for j in range(1,X):
            matriz_auxiliar[ventana[i][j]][ventana[i-1][j-1]] += 1
            normalizador += 2
    glcm = matriz_auxiliar + np.transpose(matriz_auxiliar)
    return (glcm/normalizador) if norm else glcm

def entropy(glcm_n):
    entropia = 0
    for renglon in glcm_n:
        for value in renglon:
            entropia += (value*log(value)) if (value>0) else 0
    return -entropia

def homogeneity(glcm_n):
    homogeneidad = 0
    Y, X = glcm_n.shape
    for i in range(Y):
        for j in range(X):
            homogeneidad += glcm_n[i][j]/(1+abs(i-j))
    return homogeneidad

def smoothness(glcm_n):
    ASM = 0
    Y, X = glcm_n.shape
    for i in range(Y):
        for j in range(X):
            ASM += glcm_n[i][j]**2
    return ASM

def contrast(glcm_n,k=2,n=1):
    contraste = 0
    Y, X = glcm_n.shape
    for i in range(Y):
        for j in range(X):
            contraste += ((i-j)**k)*(glcm_n[i][j]**n)
    return contraste

def imagenWEntropy(imagen,window_length,window_height,**kwargs):
    Y, X = imagen.shape
    array_aux = []
    length_overlap = (kwargs["length_overlap"] if ("length_overlap" in kwargs) else 0)
    height_overlap = (kwargs["height_overlap"] if ("height_overlap" in kwargs) else 0)
    mostrar = (kwargs["show"] if ("show" in kwargs) else False)
    for texel_renglon in range(0,Y,window_height-height_overlap):
        renglon_aux = []
        for texel_startP in range(0,X,window_length-length_overlap):
            ventana = imagen[texel_renglon:texel_renglon+window_height,texel_startP:texel_startP+window_length]
            glcm_N = glcmMaker_h(ventana,True)
            renglon_aux.append(entropy(glcm_N))
        array_aux.append(renglon_aux)
    imagen_res = np.array(array_aux)
    if mostrar:
        plt.imshow(imagen_res)
        plt.show()
    return imagen_res

def imagenWHomogeneity(imagen,window_length,window_height,**kwargs):
    Y, X = imagen.shape
    array_aux = []
    length_overlap = (kwargs["length_overlap"] if ("length_overlap" in kwargs) else 0)
    height_overlap = (kwargs["height_overlap"] if ("height_overlap" in kwargs) else 0)
    mostrar = (kwargs["show"] if ("show" in kwargs) else False)
    for texel_renglon in range(0,Y,window_height-height_overlap):
        renglon_aux = []
        for texel_startP in range(0,X,window_length-length_overlap):
            ventana = imagen[texel_renglon:texel_renglon+window_height,texel_startP:texel_startP+window_length]
            glcm_N = glcmMaker_h(ventana,True)
            renglon_aux.append(homogeneity(glcm_N))
        array_aux.append(renglon_aux)
    imagen_res = np.array(array_aux)
    if mostrar:
        plt.imshow(imagen_res)
        plt.show()
    return imagen_res


def imagenWSmoothness(imagen,window_length,window_height,**kwargs):
    Y, X = imagen.shape
    array_aux = []
    length_overlap = (kwargs["length_overlap"] if ("length_overlap" in kwargs) else 0)
```

```
    height_overlap = (kwargs["height_overlap"] if ("height_overlap" in kwargs) else 0)
    mostrar = (kwargs["show"] if ("show" in kwargs) else False)
    for texel_renglon in range(0,Y,window_height-height_overlap):
        renglon_aux = []
        for texel_startP in range(0,X,window_length-length_overlap):
            ventana = imagen[texel_renglon:texel_renglon+window_height,texel_startP:texel_startP+window_length]
            glcm_N = glcmMaker_h(ventana,True)
            renglon_aux.append(smoothness(glcm_N))
        array_aux.append(renglon_aux)
    imagen_res = np.array(array_aux)
    if mostrar:
        plt.imshow(imagen_res)
        plt.show()
    return imagen_res

def imagenWContrast(imagen,window_length,window_height,**kwargs):
    Y, X = imagen.shape
    array_aux = []
    length_overlap = (kwargs["length_overlap"] if ("length_overlap" in kwargs) else 0)
    height_overlap = (kwargs["height_overlap"] if ("height_overlap" in kwargs) else 0)
    mostrar = (kwargs["show"] if ("show" in kwargs) else False)
    for texel_renglon in range(0,Y,window_height-height_overlap):
        renglon_aux = []
        for texel_startP in range(0,X,window_length-length_overlap):
            ventana = imagen[texel_renglon:texel_renglon+window_height,texel_startP:texel_startP+window_length]
            glcm_N = glcmMaker_h(ventana,True)
            renglon_aux.append(contrast(glcm_N))
        array_aux.append(renglon_aux)
    imagen_res = np.array(array_aux)
    if mostrar:
        plt.imshow(imagen_res)
        plt.show()
    return imagen_res

def vectorizador(textura,window_l,window_h, **kwargs):
    l_over = (kwargs["length_overlap"] if ("length_overlap" in kwargs) else 0)
    h_over = (kwargs["height_overlap"] if ("height_overlap" in kwargs) else 0)
    mostrar = (kwargs["show"] if ("show" in kwargs) else False)
    entropia = imagenWEntropy(textura,window_l,window_h,length_overlap=l_over,height_overlap=h_over,show=mostrar)
    homogeneidad = imagenWHomogeneity(textura,window_l,window_h,length_overlap=l_over,height_overlap=h_over,show=mostrar)
    smoothness = imagenWSmoothness(textura,window_l,window_h,length_overlap=l_over,height_overlap=h_over,show=mostrar)
    contraste = imagenWContrast(textura,window_l,window_h,length_overlap=l_over,height_overlap=h_over,show=mostrar)
    return np.dstack((entropia,homogeneidad,smoothness,contraste))
imagenC2 = vectorizador(texturas_array[1],62,22,length_overlap=2,height_overlap=2)

# Esta funcion recibe la "imagen" compuesta de caracteristicas y devuelve la lista de "pixeles de caracteristicas"
def pixelsImage(imagen):
    pixeles_clase = []
    for renglon in imagen:
        for pixel in renglon:
            pixeles_clase.append(pixel)
    return np.array(pixeles_clase)

def mostrarColores(colores, **kwargs):
    colormap = np.zeros((100,10,3), dtype=int)
    texturemap = np.zeros((100,10,3), dtype=int)
    fig, ax = plt.subplots()
    for c in range(len(colores)):
        colormap = np.concatenate((colormap,np.full((100,100,3), colores[c]),np.zeros((100,10,3), dtype=int)), axis=1)
        ax.text(50+(110*c),50,'T'+str(c+1), style ='italic', fontsize = 15)
        if ("texturas" in kwargs):
            texturemap = np.concatenate((texturemap, np.dstack((kwargs['texturas'][c][0:100,0:100], kwargs['texturas'][c][0:100,0:100],kwargs['texturas'][c][0:100,0:100])),np.zeros((100,10,3), dtype=int)), axis=1)
    if ("texturas" in kwargs):
        colormap = np.concatenate((colormap, texturemap))
    plt.axis('off')
    plt.imshow(colormap)
    plt.show()

# Esta funci n recibe una lista de las matrices de cada clase
def trainingBayess(matrices_clases):
    # Entrenamiento
    MUs = []
    SIGMAs_inv = []
    probas = []
    ldDers = []
    colores = []
    for clase in matrices_clases:
        MU = np.mean(clase, axis=0)
        MUs.append(MU)
        SIGMA = getSigma(clase)
        SIGMA_inv = np.linalg.inv(SIGMA)
        SIGMAs_inv.append(SIGMA_inv)
        proba = probaClase(len(clase), clase.shape[0], clase.shape[1], len(matrices_clases))
        probas.append(proba)
        ldDer = ladoDerecho(np.linalg.det(SIGMA), proba)
        ldDers.append(ldDer)
        colores.append((np.random.rand(3)*1000%255).astype(int).tolist())
    mostrarColores(colores)
    return MUs, SIGMAs_inv, ldDers, colores
```

VII. Conclusión

Al haber implementado un clasificador de Bayes para la clasificación de 2 o más imágenes nos dimos cuenta de que con la aplicación directa de la regla de Bayes da como resultado una problema computacionalmente costoso que no hace más que incrementarse conforme la cantidad de imágenes de entrenamiento y la complejidad aumenta. Por tanto, si bien es un buen punto de partida, no es un clasificador óptimo. Además, aprendimos que es necesario tener un buen volumen de imágenes de entrenamiento y que para ponerlo a prueba exhaustivamente las imágenes de entrenamiento y prueba deben diferir.

VIII. Referencias

(s.a) (s.f) matplotlib.pyplot.plot Documentación de Matplotlib. Consultado de https://matplotlib.org/stable/api/_as_

gen/matplotlib.pyplot.plot.html

(s.a) (s.f) matplotlib.contour.QuadContourSet Documentación de Matplotlib. Consultado de https://matplotlib.org/stable/api/contour_api.html#matplotlib.contour.QuadContourSet

(s.a) (s.f) matplotlib.image.AxesImage Documentación de Matplotlib. Consultado de https://matplotlib.org/stable/api/image_api.html#matplotlib.image.AxesImage

(s.a) (s.f) matplotlib.pyplot.imshow Documentación de Matplotlib. Consultado de https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html

(s.a) (s.f) matplotlib.patches.Patch. Documentación de Matplotlib. Consultado de https://matplotlib.org/stable/api/_as_gen/matplotlib.patches.Patch.html

(s.a) (s.f) Shapely and geometric objects. Consultado de https://automating-gis-processes.github.io/site/notebooks/L1/geometric-objects.html

(s.a) (s.f) matplotlib.path. Documentación de Matplotlib. Consultado de https://matplotlib.org/stable/api/path$_api.html$

(s.a) (s.f) matplotlib.pyplot.plot. Documentación de Matplotlib. Consultado de https://matplotlib.org/stable/api/$_as_gen/matplotlib.pyplot.plot.html$

(s.a) (s.f) Image Resolution and DPI. Consultado de https://largeprinting.com/resources/image-resolution-and-dpi.html

(s.a)(26 de dic, 2020) Apply a Gauss filter to an image with Python. Geeks for Geeks. Consultado de https://www.geeksforgeeks.org/apply-a-gauss-filter-to-an-image-with-python/

(s.a) (14 de julio, 2019) Python PIL — GaussianBlur() method. Geeks for Geeks. Consultado de https://www.geeksforgeeks.org/python-pil-gaussianblur-method/

Banterla, D. (s/f) Texturas. Fac. Informática San Sebastián. Consultado de http://www.ehu.eus/ccwintco/uploads/d/d7/Texturas.pdf

Dabbura, I. (17 de spetiembre, 2018) K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks. Towards data science. Consultado de https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

gene (13 de abril, 2017) Geopandas Polygon to matplotlib patches Polygon conversion Stack Exchange. Consultado de https://gis.stackexchange.com/questions/197945/geopandas-polygon-to-matplotlib-patches-polygon-conversion

gene (4 de junio, 2014). Converting Matplotlib contour objects to Shapely objects. Stack Overflow. Consultado de https://gis.stackexchange.com/questions/99917/converting-matplotlib-contour-objects-to-shapely-objects

Ghandi, R. (5 de Mayo, 2018) Naive Bayes Classifier Towards Data Science. Consultado de https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

Gillies, S.(27 de sep, 2020) The Shapely User Manual. Shapely. Consultado de https://shapely.readthedocs.io/en/stable/manual.html

Hall-Beyer, M. (2017) GLCM Texture: A Tutorial v. 3.0. University of Calgary. Consultado de https://prism.ucalgary.ca/bitstream/handle/1880/51900/texture%20tutorial%20v%203_0%20180206.pdf?sequence=11&isAllowed=y

jodag. (6 de mayo, 2020) Matplotlib - unable to save image in same resolution as original image. Stack Overflow. Consultado de https://stackoverflow.com/questions/34768717/matplotlib-unable-to-save-image-in-same-resolution-as-original-image34769840

Korstanje, J. (7 de abril, 2021) The k-Nearest Neighbors (kNN) Algorithm in Python. RealPython. Consultado en https://realpython.com/knn-python/

Lin, W. et al. (2010) Image Segmentation Using the K-means Algorithm for Texture Features. World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering

Navlani, A. (2 de agosto, 2018) KNN Classification using Scikit-learn. Datacamp. Consultado en https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

R, Kirsten et al. (5 se septiembre, 2019) Performance of two multiscale texture algorithms in classifying silver gelatine paper via k-nearest neighbors. Open Archive Toulouse Archive Ouverte. Consultado de https://hal.archives-ouvertes.fr/hal-02279362/document

Rosebrock, A. (8 de agosto, 2016) k-NN classifier for image classification. pyImageSearch. Consultado en https://www.pyimagesearch.com/2016/08/08/k-nn-classifier-for-image-classification/

tom10 (23 de Marzo, 2015) Python - convert contours to image. Stack Overflow. Consultado de https://stackoverflow.com/questions/29213238/python-convert-contours-to-image29214175