

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 vectorial

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:

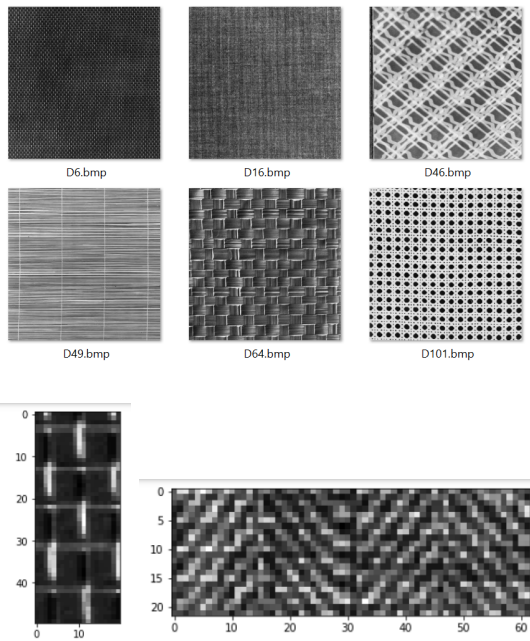
- Eficaz en espacios de gran dimensión.
- Sigue siendo eficaz en los casos en que el número de dimensiones es mayor que el número de muestras.
- Utiliza un subconjunto de puntos de entrenamiento en la función de decisión (llamados vectores de soporte), por lo que también es eficiente en la memoria.
- 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]: H
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],
               [4, 0, 6, 1],
               [0, 0, 1, 2]])

In [15]: H
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:

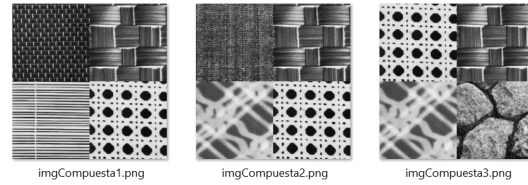
$$Entropia = \sum_{i,j=0}^{N-1} -\ln(P_{ij})P_{ij}$$

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

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

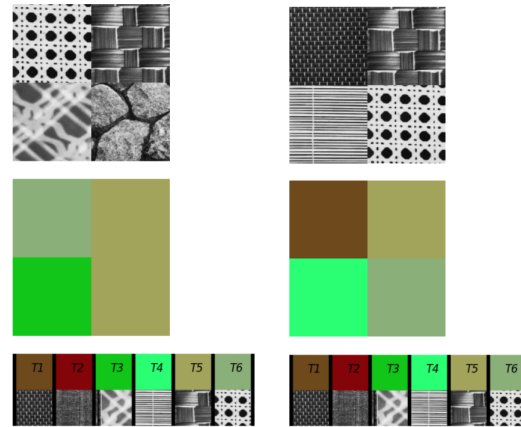
$$Contraste = \sum_{i,j=0}^{N-1} P_{ij}(i - j)^2$$

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

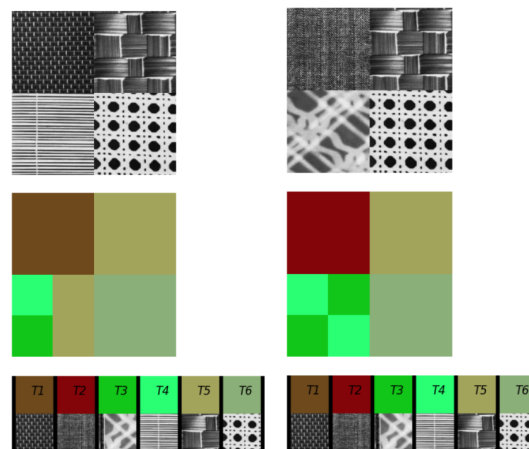


IV. RESULTADO

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. PARTE B. CLASIFICACIÓN CON SUPERPIXELES

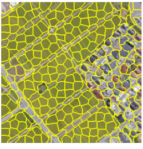
Para esta parte se busca ser capaces de clasificar imágenes ópticas satelitales de acuerdo a las texturas

La primera parte es seccionar nuestras imágenes de prueba y entrenamiento en superpíxeles

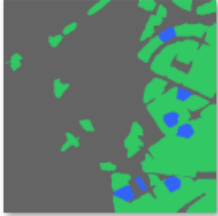
```

imagen_s1 = io.imread('satellite/pl.png')[::3]
s1_superpixel = slic(image=s1, n_segments=300, start_label=1, compactness=12.0, sigma=3.0)
plt.imshow(mark_boundaries(image=s1, s1_superpixel))
plt.axis('off')
plt.show()
plt.imsave('satellite/sp1_n300_c12_s3_'+str(np.amax(s1_superpixel))+'.png', mark_boundaries(image=s1, s1_superpixel))

```



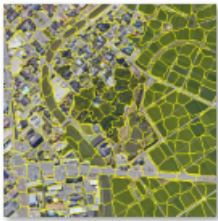
Por medio de clasificadores de K-mean y KNN obtuvimos los siguientes resultados



resultadosKmean
s.png



resultadosKNN.p
ng



sp4_n300_c12_s3
_268.png

VI. CONCLUSIONES

Esta práctica nos permitió en primera instancia a usar métodos 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 magia 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.

Para la parte B aprendimos que el uso de los megapíxeles puede llegar a ser útil para la clasificación de imágenes mucho más complejas, en nuestro caso no obtuvimos los resultados más óptimos pero aprendimos que de tener más imágenes de entrenamiento y un mejor seccionamiento en superpíxeles los resultados mejorarán. Por ende, consideramos que los objetivos de la práctica fueron cumplidos.

VII. CÓDIGO

```

1 import matplotlib.pyplot as plt
2 import cv2
3 import skimage as ski
4 import PIL as pil
5 import imageio as io
6 import numpy as np
7 from PIL import Image, ImageFilter
8 from shapely.geometry import Polygon, MultiPolygon
9 from descartes import PolygonPatch
10 from math import log
11 from sklearn.neighbors import KNeighborsClassifier
12
13 # Esta función 'desdobla' cadenas para crear un
14 # rango y un número
15 # eg: '100'=mayores que 100, '0-50'= entre 0 y 50,
16 # '10-20,500'= entre 10 y 20 o mayores de 500
17 def listRanger(rango: str):
18     # Separa la cadena por comas
19     rango = rango.split(',')
20     # Definición de las variables de retorno
21     listarango = []
22     mayorq = -1
23     # Recorre las sentencias separadas anteriormente
24     for x in rango:
25         # Si la sentencia tiene un '-' es un rango
26         if '-' in x:
27             # Los rangos se aaden a la lista
28             listarango
29             listax = x.split('-')
30             listarango += list(range(int(listax[0]),
31                                     int(listax[1])))
32             # Si no, es una cota inferior
33             else:
34                 # Solo puede existir una de estas cotas
35                 # en la sentencia
36                 mayorq = int(x)
37
38     return [listarango, mayorq]
39
40 # Esta función imprime la imagen junto con las
41 # curvas de la posición nColl de contornos
42 # Que cumplan con range y retorna un arreglo con
43 # dichas curvas.
44 # Se puede especificar los puntos en el perímetro de
45 # las líneas con range. eg: range='400-560'
46 # Si no se especifica se utiliza '150'
47 # Se puede especificar un nombre para guardar la
48 # imagen resultante con save. eg: save='imagen2.
49 # png'
50 # Si no se especifica se utiliza 'ImCrTMP.png'
51 # eg: masker.printImCr(fruta, fruta_contornos, 1,
52 # range='200', save='comida_contornos.png')
53 def printImCr(imagen, contornos, nColl, **kwargs):
54     # Desdobla range
55     rangolstd = listRanger(kwargs["range"] if ("
56     range" in kwargs) else '150')
57     # Define variable de retorno
58     curvas_array = []
59     # Con este for se muestran todas las líneas cuya
60     # primera dimensión entre en el rango
61     for i in contornos.collections[nColl].get_paths
62     ():
63         # Si cumple con las características
64         # indicadas en range se aade al plot y a la var
65         # de retorno
66         if len(i.vertices) in rangolstd[0] or ((len(
67         i.vertices) > rangolstd[1]) if (rangolstd[1] !=
68         -1) else (len(i.vertices) > len(i.vertices)+1)):
69             plt.plot(i.vertices[:,0], i.vertices
70            [:,1], '--b')
71             curvas_array.append(i.vertices)
72
73     # Muestra la imagen
74     plt.imshow(imagen)
75     plt.axis('off')
76     # Salva la imagen

```

```

57 plt.savefig('resultados/'+kwargs["save"] if ("
    save" in kwargs) else 'resultados/ImCrTMP.png',
    bbox_inches='tight', transparent=False,
    pad_inches = 0)
58 # Regresa el arreglo
59 return curvas_array
60
61 # Esta funci n imprime la imagen de fondo junto con
    los poligonos
62 # Definidos por las curvas en curvas_arr
63 # Retorna un obj Multipolygon
64 def printImPoly(curvas_arr, fondo):
65     # Arreglo aux
66     poly_array = []
67     # Recorre arreglo de curvas
68     for crvua in curvas_arr:
69         x = crvua[:,0]
70         y = crvua[:,1]
71         poly_array.append(Polygon([(i[0], i[1]) for
            i in zip(x,y)]))
72     polygons = MultiPolygon(poly_array)
73     #len(polygons.geoms)
74     #polygons
75     fig = plt.figure()
76     ax = fig.gca()
77
78     # Plotea la imagen de fondo
79     plt.imshow(fondo)
80     # Plotea los poligonos
81     for poly in polygons:
82         ax.add_patch(PolygonPatch(poly))
83     #ax.axis('scaled')
84     plt.show()
85     # Retorna los poligonos
86     return polygons
87
88 def listaclase(clase, n=4):
89     clase_array = []
90     for i in range(n):
91         objeto = io.imread('entrenamiento-procesado/
            Entrenamiento'+str(i+1)+'-'+ clase +
            '_gaussianBlur.jpg')
92         clase_array.append(np.array(objeto))
93         plt.imshow(objeto)
94         plt.show()
95     return clase_array
96
97 def pixelsMatrix(clase, n=4):
98     lista_clase = listaclase(clase, n)
99     pixeles_clase = []
100    for objeto in lista_clase:
101        for renglon in objeto:
102            for pixel in renglon:
103                if pixel.mean() > 0.:
104                    pixeles_clase.append(pixel)
105    return np.array(pixeles_clase)
106
107 def pixelsMatrix_blanco(clase, n=6):
108     lista_clase = listaclase(clase, n)
109     pixeles_clase = []
110     pixeles_probados = 0
111     for objeto in lista_clase:
112         for renglon in objeto:
113             for pixel in renglon:
114                 pixeles_probados += 1
115                 if pixel.mean() < 250.:
116                     pixeles_clase.append(pixel)
117    return np.array(pixeles_clase), pixeles_probados
118
119 def getSigma(matriz_pix):
120     MU = np.mean(matriz_pix, axis=0)
121     SIGMA = np.zeros((len(MU), len(MU)))
122
123     for pixel in matriz_pix:
124         P_MU = np.array([pixel-MU])
125         SIGMA += np.dot(np.transpose(P_MU), P_MU)
126
127     return SIGMA / len(matriz_pix)
128
129 def probaClase(lenClase, x, y, n):
130     return lenClase/(x*y*n)
131
132 def ladoDerecho(SIGMA_det, probaClase):
133     return (-log(SIGMA_det)/2) + log(probaClase)
134
135 def probaPixClase(pixel, MU, SIGMA_inv):
136     P_MU = np.array([pixel-MU])
137     #print((np.dot(np.dot(P_MU, SIGMA_inv), np.
        transpose(P_MU)) [0,0]) /-2)
138     return (np.dot(np.dot(P_MU, SIGMA_inv), np.
        transpose(P_MU)) [0,0]) /-2
139
140 def clasificador(imagen, MUs, SIGMAs_inv, ldDers,
    colores, **kwargs):
141     show = (kwargs["show"] if ("show" in kwargs)
        else False)
142     if (isinstance(imagen, str)):
143         prueba_img = io.imread(imagen)
144         prueba = np.array(prueba_img)
145     elif (isinstance(imagen, np.ndarray)):
146         prueba = imagen
147     if show:
148         plt.imshow(prueba)
149         plt.show()
150
151     dimensiones = prueba.shape
152     resultado = np.zeros((dimensiones[0], dimensiones
        [1],3), dtype=int)
153     for i in range(len(prueba)):
154         for j in range(len(prueba[i])):
155             arr_aux = []
156             for n in range(len(MUs)):
157                 arr_aux.append(probaPixClase(prueba[
                    i][j], MUs[n], SIGMAs_inv[n]) + ldDers[n])
158             resultado[i][j] = colores[arr_aux.index(
                max(arr_aux))]
159     return resultado
160
161 def makeGaussian(nombre_imagen, valor_radio = 3):
162     #codigo de internet para aplicar filtro
        Gaussiano
163     #Abriendo imagen con Image
164     image = Image.open(nombre_imagen)
165
166     #Aplicando el filtro de Gaussiano
167     #Con un valor para el radio igual a 3
168     image = image.filter(ImageFilter.GaussianBlur(
        radius = valor_radio))
169
170     #Se muestra la imagen
171     image.show()
172
173     #Se guarda esta nueva imagen borrosa
174     image.save(nombre_imagen[:-4] + "_gaussianBlur.
        jpg")
175
176 def listaTexturas(text_names):
177     texturas_array = []
178     for text_name in text_names:
179         objeto = io.imread('texturas/D'+text_name+'.
            bmp')
180         texturas_array.append(np.array(objeto)
           [:, :, 0])
181         plt.imshow(objeto)
182         plt.show()
183     return texturas_array
184
185 def windowMaker(imagen, window_length, window_height,
    **kwargs):
186     Y, X = imagen.shape
187     windows_array = []
188     length_overlap = (kwargs["length_overlap"] if ("
        length_overlap" in kwargs) else 0)
189     height_overlap = (kwargs["height_overlap"] if ("
        height_overlap" in kwargs) else 0)
190     show = (kwargs["show"] if ("show" in kwargs)

```

```

else False)
189 for texel_renglon in range(0,Y>window_height-
height_overlap):
190     for texel_startP in range(0,X>window_length-
length_overlap):
191         ventana = imagen[texel_renglon:
texel_renglon+window_height,texel_startP:
texel_startP+window_length]
192         windows_array.append(ventana)
193         if (show):
194             plt.imshow(ventana, cmap=plt.cm.gray)
)
195     plt.show()
196 return windows_array
197
198
199 def glcmMaker_v(ventana, norm = False):
200     matriz_auxiliar = np.zeros((np.amax(ventana)+1,)
*2, dtype=int)
201     Y, X = ventana.shape
202     normalizador = 0
203     for i in range(Y-1):
204         for j in range(X):
205             matriz_auxiliar[ventana[i][j]][ventana[i
+1][j]] += 1
206             normalizador += 2
207     glcm = matriz_auxiliar + np.transpose(
matriz_auxiliar)
208     return (glcm/normalizador) if norm else glcm
209
210 def glcmMaker_45(ventana, norm = False):
211     matriz_auxiliar = np.zeros((np.amax(ventana)+1,)
*2, dtype=int)
212     Y, X = ventana.shape
213     normalizador = 0
214     for i in range(1,Y):
215         for j in range(X-1):
216             matriz_auxiliar[ventana[i][j]][ventana[i
-1][j+1]] += 1
217             normalizador += 2
218     glcm = matriz_auxiliar + np.transpose(
matriz_auxiliar)
219     return (glcm/normalizador) if norm else glcm
220
221 def glcmMaker_135(ventana, norm = False):
222     matriz_auxiliar = np.zeros((np.amax(ventana)+1,)
*2, dtype=int)
223     Y, X = ventana.shape
224     normalizador = 0
225     for i in range(1,Y):
226         for j in range(1,X):
227             matriz_auxiliar[ventana[i][j]][ventana[i
-1][j-1]] += 1
228             normalizador += 2
229     glcm = matriz_auxiliar + np.transpose(
matriz_auxiliar)
230     return (glcm/normalizador) if norm else glcm
231
232 def entropy(glcm_n):
233     entropia = 0
234     for renglon in glcm_n:
235         for value in renglon:
236             entropia += (value*log(value)) if (value
>0) else 0
237     return -entropia
238
239 def homogeneity(glcm_n):
240     homogeneidad = 0
241     Y, X = glcm_n.shape
242     for i in range(Y):
243         for j in range(X):
244             homogeneidad += glcm_n[i][j]/(1+abs(i-j))
245     return homogeneidad
246
247 def smoothness(glcm_n):
248     ASM = 0
249     Y, X = glcm_n.shape
250     for i in range(Y):
251         for j in range(X):
252             ASM += glcm_n[i][j]**2
253     return ASM
254
255 def contrast(glcm_n,k=2,n=1):
256     contraste = 0
257     Y, X = glcm_n.shape
258     for i in range(Y):
259         for j in range(X):
260             contraste += ((i-j)**k)*(glcm_n[i][j]**n)
261     return contraste
262
263 def imagenWEntropy(imagen,window_length,
window_height,**kwargs):
264     Y, X = imagen.shape
265     array_aux = []
266     length_overlap = (kwargs["length_overlap"] if ("
length_overlap" in kwargs) else 0)
267     height_overlap = (kwargs["height_overlap"] if ("
height_overlap" in kwargs) else 0)
268     mostrar = (kwargs["show"] if ("show" in kwargs)
else False)
269     for texel_renglon in range(0,Y>window_height-
height_overlap):
270         renglon_aux = []
271         for texel_startP in range(0,X>window_length-
length_overlap):
272             ventana = imagen[texel_renglon:
texel_renglon+window_height,texel_startP:
texel_startP+window_length]
273             glcm_N = glcmMaker_h(ventana,True)
274             renglon_aux.append(entropy(glcm_N))
275             array_aux.append(renglon_aux)
276     imagen_res = np.array(array_aux)
277     if mostrar:
278         plt.imshow(imagen_res)
279         plt.show()
280     return imagen_res
281
282 def imagenWHomogeneity(imagen,window_length,
window_height,**kwargs):
283     Y, X = imagen.shape
284     array_aux = []
285     length_overlap = (kwargs["length_overlap"] if ("
length_overlap" in kwargs) else 0)
286     height_overlap = (kwargs["height_overlap"] if ("
height_overlap" in kwargs) else 0)
287     mostrar = (kwargs["show"] if ("show" in kwargs)
else False)
288     for texel_renglon in range(0,Y>window_height-
height_overlap):
289         renglon_aux = []
290         for texel_startP in range(0,X>window_length-
length_overlap):
291             ventana = imagen[texel_renglon:
texel_renglon+window_height,texel_startP:
texel_startP+window_length]
292             glcm_N = glcmMaker_h(ventana,True)
293             renglon_aux.append(homogeneity(glcm_N))
294             array_aux.append(renglon_aux)
295     imagen_res = np.array(array_aux)
296     if mostrar:
297         plt.imshow(imagen_res)
298         plt.show()
299     return imagen_res
300
301 def imagenWSmoothness(imagen,window_length,
window_height,**kwargs):
302     Y, X = imagen.shape
303     array_aux = []
304     length_overlap = (kwargs["length_overlap"] if ("
length_overlap" in kwargs) else 0)
305     height_overlap = (kwargs["height_overlap"] if ("

```



```

height_overlap" in kwargs) else 0)
307 mostrar = (kwargs["show"] if ("show" in kwargs)
308 else False)
for texel_renglon in range(0,Y>window_height-
height_overlap):
309     renglon_aux = []
310     for texel_startP in range(0,X>window_length-
length_overlap):
311         ventana = imagen[texel_renglon:
texel_renglon+window_height,texel_startP:
texel_startP+window_length]
312         glcm_N = glcmMaker_h(ventana,True)
313         renglon_aux.append(smoothness(glcm_N))
314         array_aux.append(renglon_aux)
315 imagen_res = np.array(array_aux)
316 if mostrar:
317     plt.imshow(imagen_res)
318     plt.show()
319 return imagen_res
320
321 def imagenWContrast(imagen>window_length,
window_height,**kwargs):
322     Y, X = imagen.shape
323     array_aux = []
324     length_overlap = (kwargs["length_overlap"] if ("
length_overlap" in kwargs) else 0)
325     height_overlap = (kwargs["height_overlap"] if ("
height_overlap" in kwargs) else 0)
326     mostrar = (kwargs["show"] if ("show" in kwargs)
327 else False)
for texel_renglon in range(0,Y>window_height-
height_overlap):
328     renglon_aux = []
329     for texel_startP in range(0,X>window_length-
length_overlap):
330         ventana = imagen[texel_renglon:
texel_renglon+window_height,texel_startP:
texel_startP+window_length]
331         glcm_N = glcmMaker_h(ventana,True)
332         renglon_aux.append(contrast(glcm_N))
333         array_aux.append(renglon_aux)
334 imagen_res = np.array(array_aux)
335 if mostrar:
336     plt.imshow(imagen_res)
337     plt.show()
338 return imagen_res
339
340 def vectorizador(textura>window_l>window_h, **kwargs
):
341     l_over = (kwargs["length_overlap"] if ("
length_overlap" in kwargs) else 0)
342     h_over = (kwargs["height_overlap"] if ("
height_overlap" in kwargs) else 0)
343     mostrar = (kwargs["show"] if ("show" in kwargs)
344 else False)
345     entropia = imagenWEntropy(textura>window_l,
>window_h,length_overlap=l_over,height_overlap=
h_over,show=mostrar)
346     homogeneidad = imagenWHomogeneity(textura,
>window_l>window_h,length_overlap=l_over,
height_overlap=h_over,show=mostrar)
347     smoothness = imagenWSmoothness(textura>window_l,
>window_h,length_overlap=l_over,height_overlap=
h_over,show=mostrar)
348     contraste = imagenWContrast(textura>window_l,
>window_h,length_overlap=l_over,height_overlap=
h_over,show=mostrar)
349     return np.dstack((entropia,homogeneidad,
smoothness,contraste))
350
351 imagenC2 = vectorizador(texturas_array[1],62,22,
length_overlap=2,height_overlap=2)
352
353 # Esta funcion recibe la "imagen" compuesta de
caracteristicas y devuelve la lista de "pixeles
de caracteristicas"
354 def pixelsImage(imagen):
355     pixeles_clase = []
356
357     for renglon in imagen:
358         for pixel in renglon:
359             pixeles_clase.append(pixel)
360     return np.array(pixeles_clase)
361
362 def mostrarColores(colores, **kwargs):
363     colormap = np.zeros((100,10,3), dtype=int)
364     texturemap = np.zeros((100,10,3), dtype=int)
365     fig, ax = plt.subplots()
366     for c in range(len(colores)):
367         colormap = np.concatenate((colormap,np.full
((100,10,3), colores[c]),np.zeros((100,10,3),
dtype=int)), axis=1)
368         ax.text(50+(110*c),50,'T'+str(c+1), style ='
italic', fontsize = 15)
369         if ("texturas" in kwargs):
370             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)
371         if ("texturas" in kwargs):
372             colormap = np.concatenate((colormap,
texturemap))
373             plt.axis('off')
374             plt.imshow(colormap)
375             plt.show()
376
377 # Esta funcion recibe una lista de las matrices de
cada clase
378 def trainingBayess(matrices_clases):
379     # Entrenamiento
380     MUs = []
381     SIGMAS_inv = []
382     probas = []
383     ldDers = []
384     colores = []
385     for clase in matrices_clases:
386         MU = np.mean(clase, axis=0)
387         MUs.append(MU)
388         SIGMA = getSigma(clase)
389         SIGMA_inv = np.linalg.inv(SIGMA)
390         SIGMAS_inv.append(SIGMA_inv)
391         proba = probaClase(len(clase), clase.shape
[0], clase.shape[1], len(matrices_clases))
392         probas.append(proba)
393         ldDer = ladoDerecho(np.linalg.det(SIGMA),
proba)
394         ldDers.append(ldDer)
395         colores.append((np.random.rand(3)*1000%255).
astype(int).tolist())
396     mostrarColores(colores)
397     return MUs, SIGMAS_inv, ldDers, colores
398
399 #Parte B
400 # GLCM horizontal
401 def sp_glcmMaker_h(imagen, marker, label_n, norm =
False):
402     matriz_auxiliar = np.zeros((np.amax(imagen)+1,)
*2, dtype=int)
403     dimensiones = imagen.shape
404     normalizador = 0
405     for i in range(dimensiones[0]):
406         for j in range(dimensiones[1]-1):
407             if (marker[i][j] == label_n and marker[i
][j+1] == label_n):
408                 matriz_auxiliar[imagen[i][j]][imagen
[i][j+1]] += 1
409                 normalizador += 2
410     glcm = matriz_auxiliar + np.transpose(
matriz_auxiliar)
411     return (glcm/normalizador) if norm else glcm
412
413 # GLCM vertical
414 def sp_glcmMaker_v(imagen, marker, label_n, norm =
False):
415     matriz_auxiliar = np.zeros((np.amax(imagen)+1,)

```

```

*2, dtype=int)
dimensiones = imagen.shape
normalizador = 0
for i in range(dimensiones[0]-1):
    for j in range(dimensiones[1]):
        if (marker[i][j] == label_n and marker[i
+1][j] == label_n):
            matriz_auxiliar[imagen[i][j]][imagen
[i+1][j]] += 1
            normalizador += 2
glcm = matriz_auxiliar + np.transpose(
matriz_auxiliar)
return (glcm/normalizador) if norm else glcm

# GLCM diagonal (esq.inf.izq -> esq.sup.der.)
def sp_glcMaker_45(imagen, marker, label_n, norm =
False):
    matriz_auxiliar = np.zeros((np.amax(imagen)+1,)
*2, dtype=int)
    dimensiones = imagen.shape
    normalizador = 0
    for i in range(1,dimensiones[0]):
        for j in range(dimensiones[1]-1):
            if (marker[i][j] == label_n and marker[i
-1][j+1] == label_n):
                matriz_auxiliar[imagen[i][j]][imagen
[i-1][j+1]] += 1
                normalizador += 2
            glcm = matriz_auxiliar + np.transpose(
matriz_auxiliar)
            return (glcm/normalizador) if norm else glcm

# GLCM diagonal (esq.sup.izq. -> esq.inf.der.)
def sp_glcMaker_135(imagen, marker, label_n, norm =
False):
    matriz_auxiliar = np.zeros((np.amax(imagen)+1,)
*2, dtype=int)
    dimensiones = imagen.shape
    normalizador = 0
    for i in range(1,dimensiones[0]):
        for j in range(1,dimensiones[1]):
            if (marker[i][j] == label_n and marker[i
-1][j-1] == label_n):
                matriz_auxiliar[imagen[i][j]][imagen
[i-1][j-1]] += 1
                normalizador += 2
            glcm = matriz_auxiliar + np.transpose(
matriz_auxiliar)
            return (glcm/normalizador) if norm else glcm

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

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

#Angular Second Moment, Energy
def smoothness(glc_n):
    ASM = 0
    Y, X = glc_n.shape
    for i in range(Y):
        for j in range(X):
            ASM += glc_n[i][j]**2
    return ASM

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

def sp_glcms(imagen, marker):
    vcs_glc = []
    for nSP in np.unique(marker):
        glcm_h = sp_glcMaker_h(rgb2gray(imagen),
marker, nSP, True)
        glcm_v = sp_glcMaker_v(rgb2gray(imagen),
marker, nSP, True)
        vcs_glc.append([entropy(glc_h), smoothness
(glc_h)])
    return vcs_glc

def rgb2gray(imagen):
    gris = ((imagen[:, :, 0]+imagen[:, :, 1]+imagen
[:, :, 2])/3).astype(int)
    return gris

def mostrarSP(superpíxeles, clasi, colores, imagen,
guardar='aux'):
    superpixel_aux = np.zeros(imagen.shape, dtype=np
.uint8)
    Y,X = superpíxeles.shape
    for i in range(Y):
        for j in range(X):
            superpixel_aux[i,j] = colores[clasi[
superpíxeles[i,j]-1]]
    plt.imshow(superpixel_aux)
    plt.axis('off')
    plt.show()
    plt.imsave('satelite/resultados'+guardar+'.png'
, superpixel_aux)

```

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 setiembre, 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 de 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>