Práctica 3. Caracterización de Texturas Brodatz

Universidad Nacional Autónoma de México
Facultad de Ingeniería
Reconocimiento de Patrones (757-1)
Basile Álvarez Andrés José
Keller Ascencio Rodolfo Andrés
Shen Shuai
*

***Resumen*—A lo largo de este documento se presentarán los resultados y análisis de nuestra tercera práctica de la materia de Reconocimiento de Patrones, donde se estará trabajando con distintos métodos de clasificación de texturas dentro de imágenes a partir del uso de la Matriz GLCM así como de los Clasificadores Bayesiano y K-NN en el lenguaje de programación Python, evaluando su precisión y capacidad de clasificación. El objetivo principal de esta práctica busca lograr clasificar imágenes con 4 regiones utilizando el Clasificadores Bayesiano y K-NN a partir de un vector de características obtenido mediante el uso de la matriz GLCM.**

I. INTRODUCCIÓN

I-A. GLCM: Grey Level Co-ocurrence Matrix

La Matriz GLCM (*Grey Level Co-ocurrence Matrix*, también conocida como *Grey Tone Spatial Dependency Matrix*) así como las métricas de texturas son estadísticas descriptivas que nos permiten la caracterización de texturas, donde la Matriz GLCM es una tabulación que indica qué tan a menudo ocurren diferentes combinaciones de valores de brillo de píxeles (niveles de gris) en una imagen. Esta idea proviene de una serie de documentos realizados por Robert Haralick y otros co-autores en la década de 1970.[1]

Para poder realizar la caracterización de texturas existe una serie de texturas estándar llamadas Texturas Brodatz en las que se prueban los algoritmos de texturizado para trabajar métodos de clasificación, las cuales incluyen imágenes como cuero, hierba, tejido de tela, fieltro y similares. Con este objetivo en mente, Haralick propuso catorce medidas distintas para el estudio de texturas.[1]

I-B. Clasificador Bayesiano

Los métodos de Bayes ingenuo representan un conjunto de algoritmos de aprendizaje supervisado que se basan en la aplicación del teorema de Bayes, asumiendo la *ingenua* condición de que las características son independientes entre sí, facilitando el cálculo de las probabilidades de cada clase. Los clasificadores Bayesianos son utilizados en gran medida para aplicaciones de clasificación de texto, minería de datos e incluso para el reconocimiento de objetos en imágenes (como se verá a lo largo de esta práctica), buscando categorizar una muestra en una o varias clases. Sin embargo, a diferencia de los clasificadores discriminantes (como la regresión logística), el clasificador Bayesiano no aprende cuáles son las características más importantes para distinguir entre las clases. [2]

La formulación del clasificador Bayesiano se basa en utilizar la regla de Bayes para calcular la probabilidad *a posteriori* de la clase dados los atributos, como se muestra a continuación: [3]

$$P(C_k|x_1,...,x_n) = \frac{P(C_k)P(x_1,...,x_n|C_k)}{P(x_1,...,x_n)} \tag{1}$$

El clasificador Bayesiano, entonces, consiste en asignar un objeto descrito por un conjunto de atributos o características x_1, ... , x_n, a una de las *k* clases posibles, C_1, ... , C_k, tal que la probabilidad de la clase dados los atributos se maximice:

$$ArgMax(P(C_k|x_1,...,x_n)) \tag{2}$$

Debido a que el denominador de (1) no varía para las diferentes clases, se puede simplificar la fórmula considerándolo como una constante:

$$P(C_k|x_1,...,x_n) = \alpha P(C_k)P(x_1,...,x_n|C_k) \tag{3}$$

El clasificador Bayesiano se basa en la suposición de que todos los atributos o características son independientes dada la clase, o sea que cada x_i es condicionalmente independiente de los demás atributos dada la clase:

$$P(x_i|x_j,C) = P(x_i|C_k), \forall i \neq j \tag{4}$$

Por lo que podemos reescribir (1) como:

$$P(C_k|x_1,...,x_n) = \frac{1}{Z}\prod_{i=1}^{n} P(C_k)P(x_i|C_k) \tag{5}$$

I-C. Clasificador K-NN

El Método *K-Nearest Neighbours* o en español K-Vecinos más cercanos pertenece al grupo de métodos de clasificación de datos, siendo un método basado en casos o instancias el cual consiste en extraer información de un conjunto de datos conocidos y usarla para clasificar nuevos datos o para agrupar datos existentes. En este sentido, un concepto muy importante a tomar en cuenta es la heurística de consistencia el cual se refiere a que en un determinado momento cuando

se desea adivinar una propiedad de algo, sin disponer de más información que un conjunto de casos de referencia, se busca hallar el caso más parecido con respecto a las propiedades conocidas para conocer la propiedad deseada.[4]

Los métodos basados en vecindad son fundamentalmente dependientes de la distancia y, en consecuencia, poseen caracteristicas relacionadas con la cercanía, la lejanía y la magnitud de longitud, entre otras.[4]

I-D. Conjunto de datos utilizado

Para la realización de esta práctica, se decidió trabajar con un conjuntos de imágenes de texturas obtenidas de Brodcatz, donde dichas texturas fueron divididas en ventanas, siendo que el 80 % de las ventanas de cada imagen de texturas fueron utilizados para el conjunto de entrenamiento y un 20 % para un conjunto de prueba. Todas las imágenes en este conjunto de datos se encuentran en formato de imagen BMP, con un tamaño de 640x640px.[5]

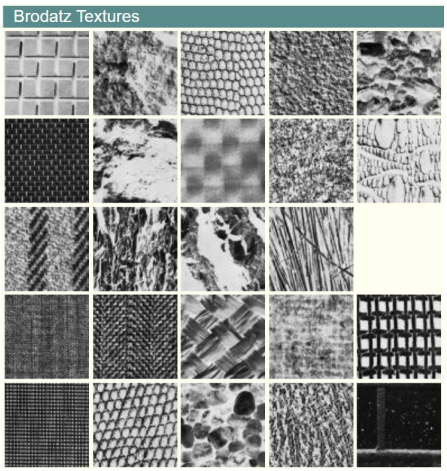


Figura 1: Ejemplo de algunas de las imágenes pertenecientes al conjunto de texturas de Brodatz.

El segundo conjunto de imágenes utilizado para comprobar el correcto funcionamiento del algoritmo se encuentra conformado por tres imágenes compuestas por una combinación de imágenes contenidas en el conjunto de texturas de Brodatz. Todas las imágenes de este conjunto de datos se encuentran en formato PNG, con un tamaño de 320x320px.[5]

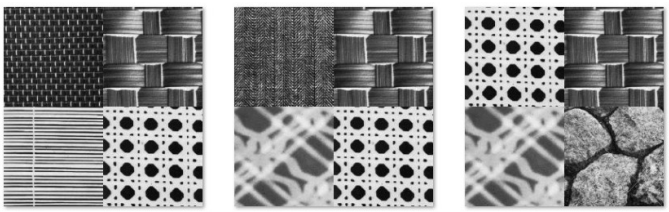


Figura 2: Imágenes pertenecientes al conjunto de prueba del funcionamiento de nuestro caracterizador de texturas.

I-E. Herramientas utilizadas

Para el desarrollo de esta práctica, se decidió trabajar con el lenguaje de programación Python realizando la lectura de imágenes de siete texturas obtenidas de Brodatz para calcular su matriz GLCM y de esta manera generar un vector de características para distinas ventanas de las imágenes de texturas. Se hizo uso de bibliotecas para implementar los Clasificadores Bayesianos y K-NN para poder realizar una clasificación de futuras imágenes compuestas a partir. Los específicos de la implementación utilizando el lenguaje Python serán vistos a lo largo del desarrollo y resultados de la práctica, así como en el apartado V de este documento.

II. OBJETIVO

El objetivo principal de esta práctica se enfoca en que el alumno logre desarrollar métodos de caracterización de texturas así como que el alumno aprenda a utilizar clasificadores como el Bayesiano, K-NN, K-Means o Máquinas de Soporte Vectorial.

III. DESARROLLO

Como fue mencionado anteriormente, la implementación del clasificador Bayesiano se realizó para clasificar imágenes de dos conjuntos de datos distintos: el primero de ellos de un conjunto de texturas obtenido de Bodatz, en donde se buscaba reconocer y entrenar modelos a partir de imágenes de texturas; el segundo conjunto donde se implementó el Clasificador Bayesiano y K-NN consiste entrenado en imágenes de texturas compuestas, donde el propósito se encontraba en determinar las regiones pertenecientes a cada una de las texturas cargadas en los modelos, donde de esta forma lográramos determinar la localización dentro de la imagen en la cual se encuentran las diversas texturas cargadas en el programa.

III-A. Selección y carga de imágenes de Texturas de Brodatz

Para poder trabajar de manera correcta con nuestro caracterizador de texturas, primero fue necesario cargar las imágenes de textura seleccionadas de Brodatz a nuestro programa. Para poder mejorar los resultados obtenidos es importante realizar un análisis de nuestras imágenes, donde se apreció que todas las imágenes tenían el mismo tamaño, siendo este de 640x640px.

Para la práctica elegimos las imágenes de texturas D16.bmp, D64.bmp, D46.bmp, D101.bmp, D6.bmp, D49.bmp y Piedras.jpg, las cuales fueron inicialmente manipuladas a través del uso de la biblioteca *Python Imaging Library* (PIL).

III-B. Función de división de imágenes en ventanas

Tras la carga de imágenes es necesario realizar una función que nos permita seccionar una imagen en ventanas, las cuales serían almacenadas en una lista por cada imagen seccionada. Lo que se hizo fue nuevamente cargar la imagen con ayuda de la biblioteca *Python Imaging Library* (PIL), cambiar la imagen para que manejara únicamente tonos dentro de la escala de grises, e iterar sobre la imagen a un determinado valor de

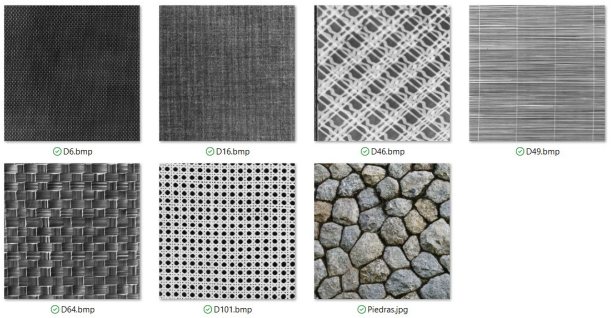


Figura 3: Conjunto de texturas seleccionadas para el entrenamiento de nuestro modelo caracterizador de texturas.

pixeles de desplazamiento elegido por el usuario para crear ventanas de pixeles de un tamaño también determinado por el usuario. Uno de los problemas que enfrentamos al momento de realizar este proceso es que al momento de calcular las características de cada ventana, no se toma en cuenta los bordes de la ventana, siendo que únicamente se considera el pixel central.

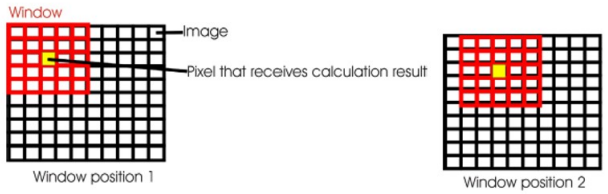


Figura 4: Esquema que indica el proceso de división y creación de ventanas dentro de una imagen.[1]

Para el caso de la práctica decidimos hacer uso de tamaños de ventana de 51x51px. con desplazamientos de 5 pixeles, obteniendo 13924 ventanas por cada imagen. El valor del tamaño de ventana y desplazamiento puede variar dentro de nuestro código, sin embargo, al trabajar con estos valores obtuvimos buenos resultados al momento de realizar la caracterización de texturas.

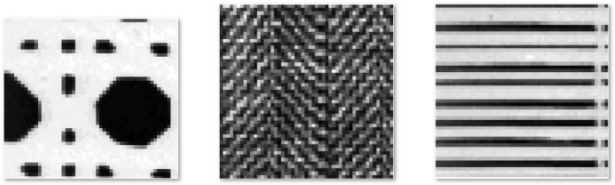


Figura 5: Ejemplos de ventanas creadas dentro de una imagen con tamaños de 51x51px.

III-C. Función para obtención de Matriz GLCM y estadísticos de segundo orden

Tras haber obtenido por cada imagen una lista de ventanas se procedió a crear una función para determinar la Matriz GLCM por cada ventana. Hicimos uso de los módulos *graycomatrix* y *graycoprops* de la biblioteca *skimage.features* para la creación de esta matriz.

Las características de nuestra matriz debían poder modificarse a partir de las necesidades del usuario, siendo estas la distancia entre los pixeles vecinos y el valor del ángulo de desplazamiento entre éstos. Para el caso específico de esta práctica trabajamos con dos casos, el primero contaba con una distancia entre pixeles vecinos de 1, con un ángulo de 0º; para el segundo caso se modificó la distancia a 2, con un ángulo de 90º.[1]

Dentro de los estadísticos de segundo grado a utilizar para la creación del vector de características de cada ventana se decidió trabajar con contraste, correlación, homogeneidad, energía, media y entropía. A estos vectores de características se les añadió una etiqueta que contuviera el nombre de la textura a la que pertenecía la ventana para la que se calcularon los estadísticos de segundo grado.

	contrast	correlation	homogeneity	energy	mean	entropy
0	932.284314	0.819458	0.043603	0.021126	0.03891	-339.548909
1	956.120784	0.830434	0.043694	0.021104	0.03891	-334.794022
2	956.105882	0.845562	0.044083	0.021140	0.03891	-342.303797
3	947.888627	0.859696	0.044698	0.021067	0.03891	-327.284247
4	934.290196	0.874386	0.044134	0.021060	0.03891	-324.039135
...	...	...	...	...	...	...
14156	1133.306275	0.573375	0.035879	0.021700	0.03891	-479.097784
14157	1163.945098	0.556625	0.035577	0.021863	0.03891	-518.872221
14158	1201.915294	0.542158	0.036037	0.021834	0.03891	-512.117333
14159	1192.633333	0.571737	0.036876	0.021799	0.03891	-506.833121
14160	1203.730980	0.584071	0.034292	0.021757	0.03891	-496.607558

Figura 6: Ejemplo de una lista que contiene los vectores de características de las ventanas de una imagen.

Para poder proceder a la implementación de los clasificadores fue necesario unir cada una de las listas que contenían los vectores de características de las ventanas de cada imagen, creando un DataFrame que contuviera toda la información de nuestras imágenes de Brodatz.

III-D. Implementación de Clasificador Bayesiano

El primer clasificador que implementamos fue el Clasificador Bayesiano, donde para su implementación se tuvo que dividir el conjunto de datos en dos *DataFrames* que contuvieran a los vectores de características por un lado, y a la textura por otro, siendo esta la característica a predecir.

Para el entrenamiento y la evaluación de nuestro modelo se trabajó con un conjunto de entrenamiento del 80 % y un conjunto de prueba del 20 % del total de ventanas que teníamos.

Este procedimiento se realizó en dos ocasiones, uno para los vectores de características obtenidas con una distancia entre pixeles vecinos de 1 y con un ángulo de 0º, así como una segunda ocasión para el segundo caso que trabajaba con una distancia de 2 y con un ángulo de 90º, obteniendo una precisión del 97.08 % y 99.39 %, respectivamente.

III-E. Implementación de Clasificador KNN

El segundo clasificador que implementamos fue el Clasificador KNN, en el cual se realizó prácticamente el mismo proceso

```
nb_01 = Naive_Bayes(df_final_01)
```

```
Score: 0.97088173583747
Matriz de confusión:
 [[2813    0    0    0    0    0    0]
 [   0 2784    0    0    0    0    0]
 [   0    0 2499  270    5   57    0]
 [   0    0   71 2769    0    0    0]
 [   0    0    0    0 2736    0    0]
 [   0    0  155    0    0 2627    0]
 [   0    0    0    0    0   11 2744]]
```

```
nb_290 = Naive_Bayes(df_final_290)
```

```
Score: 0.993910240008188
Matriz de confusión:
 [[2813    0    0    0    0    0    0]
 [   0 2782    0    0    0    2    0]
 [   0    0 2786    0    3   42    0]
 [   0    0    0 2840    0    0    0]
 [   0    0    0    0 2736    0    0]
 [   0   15   36    0    2 2729    0]
 [   0    0    3    0    0   16 2736]]
```

Figura 7: Ejemplo de una lista que contiene los vectores de características de las ventanas de una imagen.

```
knn_01 = KNN(df_final_01)
```

```
Puntaje: 0.9893045391740443
Matriz de confusión:
 [[2813    0    0    0    0    0    0]
 [   0 2784    0    0    0    0    0]
 [   0    0 2737   30    1   63    0]
 [   0    0   65 2775    0    0    0]
 [   0    0    0    0 2736    0    0]
 [   0    0   50    0    0 2732    0]
 [   0    0    0    0    0    0 2755]]
```

```
knn_290 = KNN(df_final_290)
```

```
Puntaje: 0.9933473210173481
Matriz de confusión:
 [[2813    0    0    0    0    0    0]
 [   0 2774    0    0    0   10    0]
 [   0    0 2767    0    0   64    0]
 [   0    0    0 2840    0    0    0]
 [   0    0    0    0 2736    0    0]
 [   0   10   46    0    0 2726    0]
 [   0    0    0    0    0    0 2755]]
```

Figura 8: Ejemplo de una lista que contiene los vectores de características de las ventanas de una imagen.

que para la implementación del Clasificador Bayesiano, con la diferencia de que en este caso también se tuvo que indicar el número de vecinos con el que se quería trabajar, donde nosotros decidimos trabajar con tres vecinos.

Al igual que en el caso anterior se dividió el conjunto de datos en dos *DataFrames* que contuvieran a los vectores de características por un lado, y a la textura por otro, siendo esta la característica a predecir.

Para el entrenamiento y la evaluación de nuestro modelo se trabajó nuevamente con un conjunto de entrenamiento del 80 % y un conjunto de prueba del 20 % del total de ventanas que teníamos.

Este procedimiento también se realizó en dos ocasiones, uno para los vectores de características obtenidas con una distancia entre pixeles vecinos de 1 y con un ángulo de 0º, y una segunda ocasión para el segundo caso que trabajaba con una distancia de 2 y con un ángulo de 90º, obteniendo una precisión del 98.93 % y 99.33 %, respectivamente.

III-F. Caracterizador de texturas a partir de imágenes combinadas

Tras haber creado cuatro clasificadores con diferentes características, dos bayesianos y dos knn, se procedió con el proceso de caracterización de texturas de las tres imágenes combinadas con las que contábamos.

Para esto, decidimos crear una función en la cual indicáramos el nombre del archivo que deseábamos caracterizar, el tamaño de ventanas, el valor de desplazamiento de pixeles y el método mediante el cual se deseaba realizar la caracterización recordando que contamos con cuatro métodos:

1. Clasificador Bayesiano con distancia entre pixeles vecinos de 1, ángulo de 0º, precisión del 97.08 %.
2. Clasificador Bayesiano con distancia entre pixeles vecinos de 2, ángulo de 90º, precisión del 99.39 %.
3. Clasificador KNN con distancia entre pixeles vecinos de 1, ángulo de 0º, precisión del 98.93 %.
4. Clasificador KNN con distancia entre pixeles vecinos de 2, ángulo de 90º, precisión del 99.33 %.

En la siguiente sección se apreciarán los resultados obtenidos con nuestro método de caracterización de texturas.

IV. RESULTADOS

Tras haber realizado el entrenamiento de nuestros cuatro modelos de caracterización de texturas pudimos poner a prueba nuestras imágenes compuestas a través de la predicción y clasificación con estos modelos.

Para los resultados de la caracterización de texturas dentro de nuestras imágenes compuestas, a partir de la predicción realizada por el clasificador Bayesiano, creamos una nueva imagen donde coloreamos cada uno de los pixeles de dichas imágenes compuestas con dimensiones 320x320px según la clase a la que pertenece. En este sentido contábamos con siete clases donde para la clase perteneciente a la textura D101 se decidió usar el color rojo, para la textura con nombre D64 se decidió el verde, la textura D46 se trabajó con amarillo, la textura D16 con azul, la textura D6 con gris, la textura D49 con rosa y finalmente la textura Piedras con morado. De esta manera coloreamos cada uno de los pixeles a partir de la clasificación que obtuvimos de los modelos Bayesianos y KNN.

A partir de las imágenes generadas, a simple vista logramos apreciar que las caracterizaciones realizadas con una distancia de un pixel y 0º grados fueron mucho más precisas que las realizadas con una distancia de dos pixeles y 90º. Por otro lado, otra de las observaciones que apreciamos fue el hecho de que las áreas cercanas al cambio de texturas resultaban contar con la mayor cantidad de errores de clasificación, donde consideramos que estos errores se deben al cálculo de la matriz GLCM de las ventanas que pasan por las regiones donde se combinan estas texturas. Estos errores se podrían ver reducidos al trabajar con tamaños de ventanas menores así como con menores distancias de desplazamiento, sin embargo, al realizar este proceso podíamos observar que el tiempo de ejecución del código aumentaba significativamente, por lo cual decidimos trabajar finalmente con tamaños de ventana de 51x51px y desplazamientos de 5 pixeles obteniendo los resultados que se apreciarán a continuación.

Para el caso de la caracterización de la primer imagen compuesta, al trabajar con el Clasificador Bayesiano, una distancia de dos pixeles y un ángulo de 90º grados se aprecia un gran error de clasificación, obteniendo un resultado con una precisión del 21.91 %, lo mismo sucede en menor medida al hacer uso del Clasificador KNN con una distancia de dos pixeles y un ángulo de 90º grados, donde se obtuvo un resultado con una precisión del 44.54 %. A partir de estos resultados podemos reconocer la importancia de una correcta selección de parámetros para trabajar con estos métodos de caracterización tomando en consideración los tiempos de ejecución necesarios para trabajar con nuestro código.

Como se mencionó en el apartado anterior, se midieron las métricas de desempeño de los modelos obtenidos, sin embargo, dichas métricas varían al momento de trabajarse en un entorno de imagen compuesta debido al manejo y cálculo de valores con los que se trabajan las imágenes, por lo que es necesario realizar nuevas métricas para el caso de las imágenes con las que trabajamos.

A manera de ejemplo enlistamos los valores de precisión obtenidos al caracterizar las texturas de la primera imagen compuesta con cada uno de nuestros cuatro modelos de clasificación entrenados.

1. Clasificador Bayesiano con distancia entre pixeles vecinos de 1, ángulo de 0º. Precisión de clasificación imagen compuesta 01: 88.03 %.
2. Clasificador Bayesiano con distancia entre pixeles vecinos de 2, ángulo de 90º. Precisión de clasificación imagen compuesta 01: 21.91 %
3. Clasificador KNN con distancia entre pixeles vecinos de 1, ángulo de 0º. Precisión de clasificación imagen compuesta 01: 83.43 %
4. Clasificador KNN con distancia entre pixeles vecinos de 2, ángulo de 90º. Precisión de clasificación imagen compuesta 01: 44.54 %

Comparando ahora los resultados obtenidos al momento de trabajar la caracterización de nuestras tres imágenes compuestas a partir del uso de nuestro tercer modelo de predicción, es decir, el Clasificador KNN con distancia entre pixeles vecinos de 1, ángulo de 0º:

- Imagen Compuesta 01: Clasificador KNN con distancia entre pixeles vecinos de 1, ángulo de 0º. Precisión de clasificación - 83.43 %
- Imagen Compuesta 02: Clasificador KNN con distancia entre pixeles vecinos de 1, ángulo de 0º. Precisión de clasificación - 85.59 %
- Imagen Compuesta 03: Clasificador KNN con distancia entre pixeles vecinos de 1, ángulo de 0º. Precisión de clasificación - 86.04 %

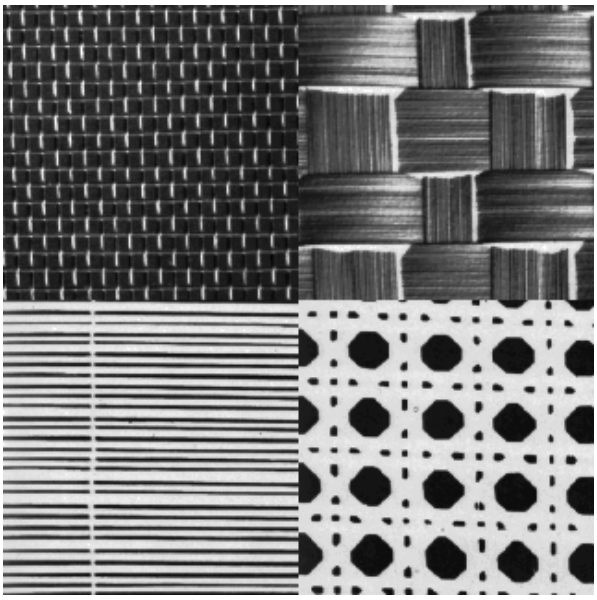
Como se puede apreciar, los valores de precisión son bastante aceptables tomando en cuenta que los tamaños de ventana elegidos fueron de 51x51px y un desplazamiento de 5 pixeles. Estos parámetros se pueden modificar para obtener precisiones mayores, sin embargo, los tiempos de entrenamiento y ejecución también aumentarán, por lo que consideramos que para fines prácticos estos parámetros cumplían con el objetivo de la práctica.

A manera de comentario, a lo largo de la realización de la práctica se estuvo trabajando con parámetros como tamaños de ventana de 101x101px con un desplazamiento de 10 pixeles, 101x101px con un desplazamiento de 1 pixel, 31x31px con un desplazamiento de 10 pixeles, 31x31px con un desplazamiento de 5 pixeles y finalmente el mejor resultado obtenido con un tiempo de ejecución bastante rápido y aceptable fue trabajando con 51x51px y un desplazamiento de 5 pixeles, parámetros utilizados para reportar los resultados de esta práctica.

A continuación mostraremos algunos de los resultados obtenidos al trabajar con la ejecución de nuestro código.

IV-A. Imagen Compuesta 01

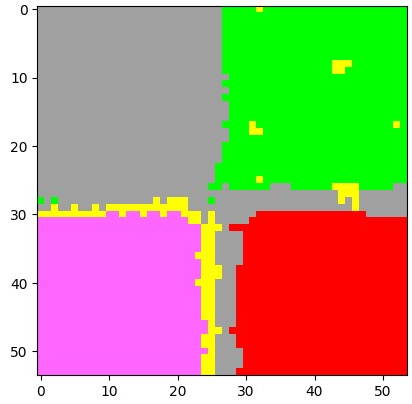
Para el caso de la primera imagen compuesta decidimos caracterizar las texturas de la imagen a través de los cuatro métodos de clasificación.



(a)

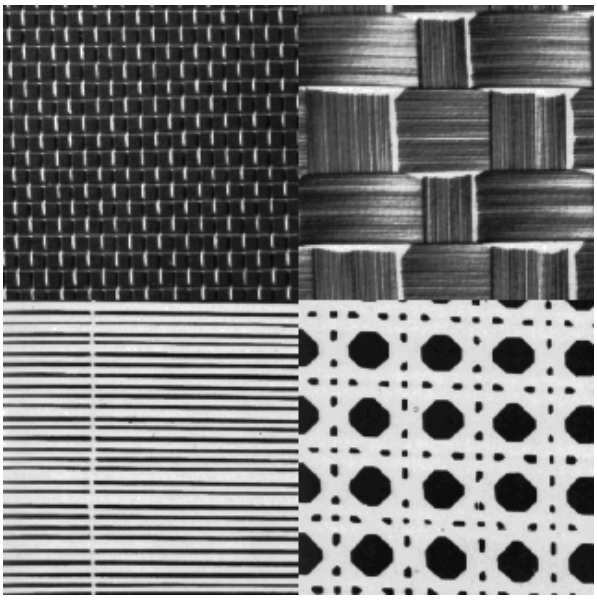
```
D6.bmp      958
D64.bmp     709
D101.bmp    583
D49.bmp     559
D46.bmp     107
Name: clase, dtype: int64
```

```
mostrar_Clasificacion(features_Compuesta_01)
```



(b)

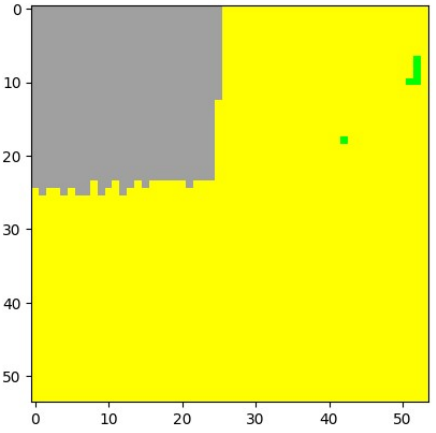
Figura 9: Imagen compuesta de prueba (a) e imagen resultado del Clasificador Bayesiano con distancia entre pixeles vecinos de 1, ángulo de 0º, precisión de la clasificación 88.03 % (b).



(a)

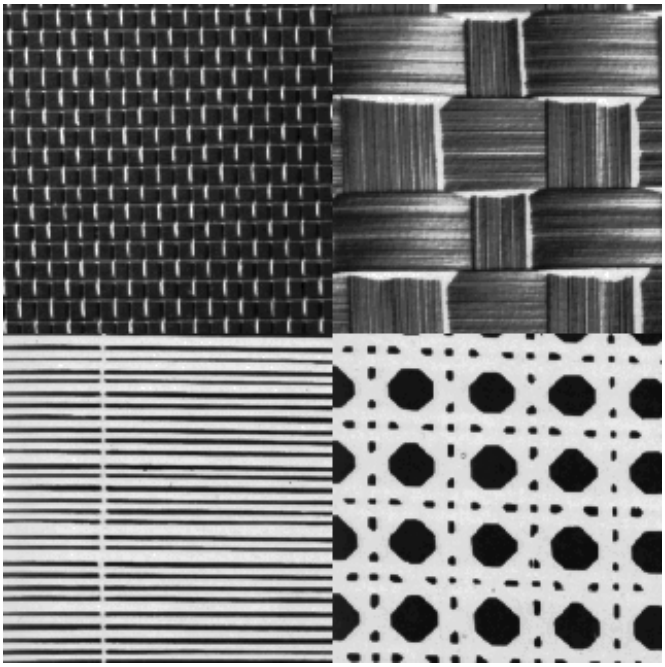
```
D46.bmp    2277
D6.bmp      633
D64.bmp       6
Name: clase, dtype: int64
```

```
mostrar_Clasificacion(features_Compuesta_02)
```



(b)

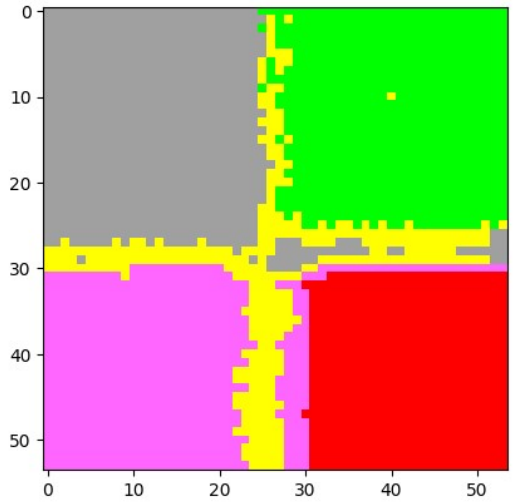
Figura 10: Imagen compuesta de prueba (a) e imagen resultado del Clasificador Bayesiano con distancia entre pixeles vecinos de 2, ángulo de 90º, precisión de la clasificación 21.91 % (b).



(a)

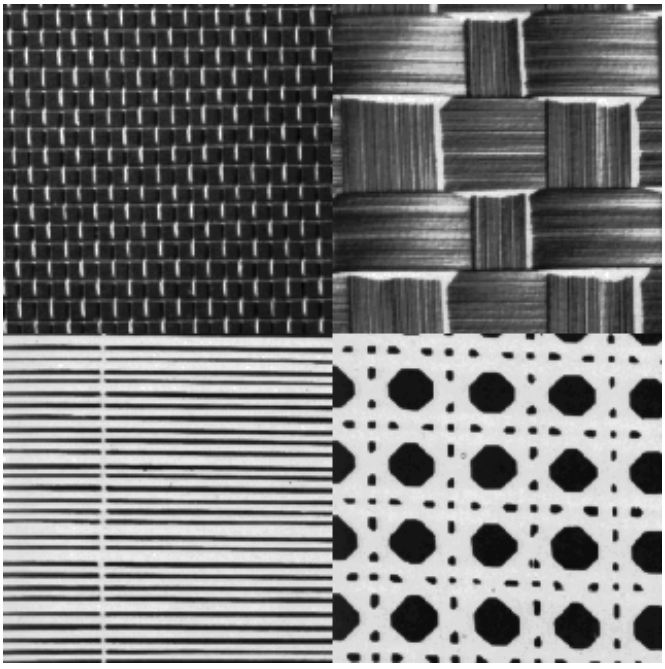
```
D6.bmp      752
D64.bmp     677
D49.bmp     634
D101.bmp    529
D46.bmp     324
Name: clase, dtype: int64
```

```
mostrar_Clasificacion(features_Compuesta_03)
```



(b)

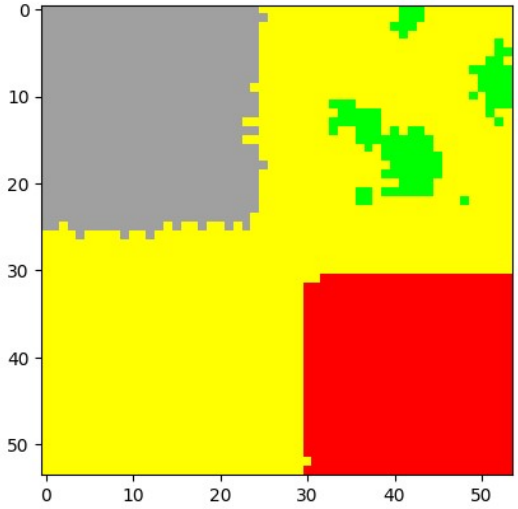
Figura 11: Imagen compuesta de prueba (a) e imagen resultado del Clasificador KNN con distancia entre pixeles vecinos de 1, ángulo de 0º, precisión de la clasificación 83.43 % (b).



(a)

```
D46.bmp     1617
D6.bmp       641
D101.bmp     549
D64.bmp      109
Name: clase, dtype: int64
```

```
mostrar_Clasificacion(features_Compuesta_04)
```

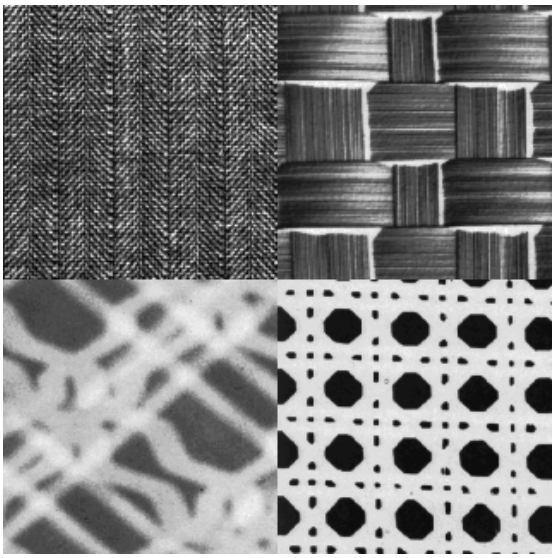


(b)

Figura 12: Imagen compuesta de prueba (a) e imagen resultado del Clasificador KNN con distancia entre pixeles vecinos de 2, ángulo de 90º, precisión de la clasificación 44.54 % (b).

IV-B. Imagen Compuesta 02

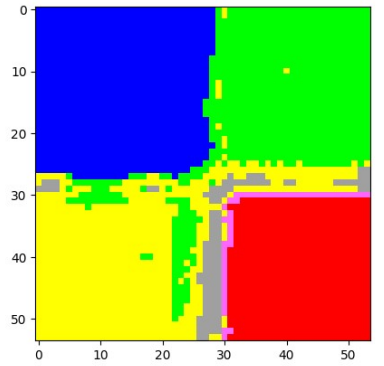
Para el caso de la segunda imagen compuesta decidimos caracterizar las texturas de la imagen a través del tercer método de clasificación, siendo este el Clasificador KNN con una distancia de 1 y un ángulo de 0º, siendo este con el que obtuvimos el mejor resultado en la primera imagen compuesta.



(a)

```
D16.bmp     768
D64.bmp     762
D46.bmp     712
D101.bmp    520
D6.bmp      107
D49.bmp      47
Name: clase, dtype: int64
```

```
mostrar_Clasificacion(features_Compuesta_01)
```

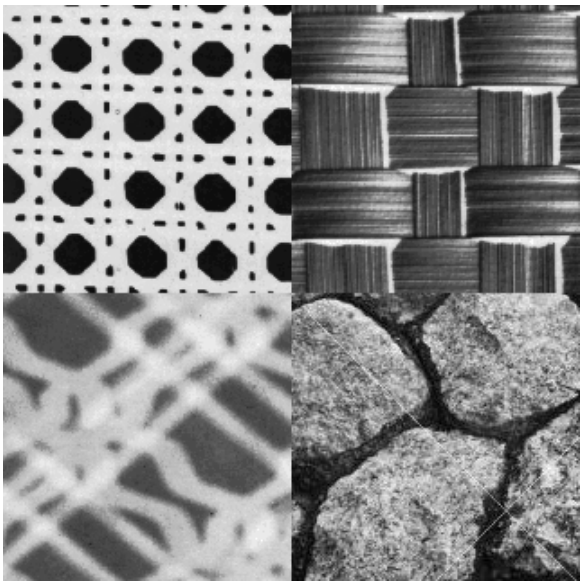


(b)

Figura 13: Imagen compuesta de prueba (a) e imagen resultado del Clasificador KNN con distancia entre pixeles vecinos de 1, ángulo de 0º, precisión de la clasificación 85.59 % (b).

IV-C. Imagen Compuesta 03

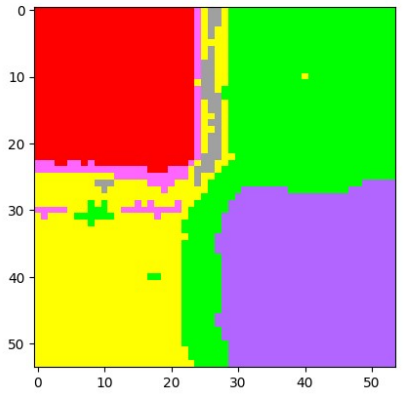
Para el caso de la tercera y última imagen compuesta decidimos caracterizar las texturas de la imagen nuevamente a través del tercer método de clasificación, siendo este el Clasificador KNN con una distancia de 1 y un ángulo de 0º, debido a los resultados obtenidos con anterioridad.



(a)

```
D64.bmp         851
Piedras.jpg     692
D46.bmp         666
D101.bmp        572
D49.bmp          84
D6.bmp           51
Name: clase, dtype: int64
```

```
mostrar_Clasificacion(features_Compuesta_01)
```



(b)

Figura 14: Imagen compuesta de prueba (a) e imagen resultado del Clasificador KNN con distancia entre pixeles vecinos de 1, ángulo de 0º, precisión de la clasificación 86.04 % (b).

V. CÓDIGO

Durante el desarrollo de esta práctica, se utilizó Google Colab como herramienta de gestión de versiones del código, así como método para trabajar de forma remota con el equipo. Se presentan algunos elementos importantes del código a continuación:

V-A. Función para lectura y despliegue de imágenes de textura

```
### Lectura de imágenes seleccionadas dentro del folder Brodatz_examples
dir_folder = "/gdrive/MyDrive/Basile_Keller_Shen_Practica3/"
files_name = ['D16.bmp','D64.bmp','D46.bmp','D101.bmp','D6.bmp','D49.bmp','Piedras.jpg']
count = 0
for img in files_name:
  print(img+':')
  img = PIL.Image.open(dir_folder + "Brodatz_examples/" + img)
  display(img)
```

V-B. Función para la división y creación de ventanas de las imágenes

```
from skimage import io
import numpy as np

def divide_windows(file, win_size, desplazamiento):
  img = PIL.Image.open(dir_folder + file)
  display(img)
  #Imagen en escala de grises
  img = io.imread(dir_folder+file, as_gray=True)
  #Nivel de escala de gris discreto
  img_gray = np.floor(img * 255).astype(np.uint8)

  #Lista que almacena las ventanas
  windows = []

  #Iteramos sobre las filas y columnas de la imagen con un desplazamiento definido por el usuario.
  for r in range(0,img_gray.shape[0] - win_size + 1, desplazamiento):
      for c in range(0,img_gray.shape[1] - win_size + 1, desplazamiento):
          #Se crea una ventana del tamaño de pixeles elegidos por el usuario a partir de la posición actual
          window = img_gray[r:r+win_size, c:c+win_size]
          windows.append(window)
  #Mostramos la cantidad de ventanas generadas
  print("Con ventanas de "+str(win_size)+"x"+str(win_size)+" Obtuvimos: "+str(len(windows))+" ventanas")
  return windows
```

V-C. Función para la generación de la matriz GLCM

```
import numpy as np
from skimage.feature import graycomatrix, graycoprops

def get_GLCM(distance, angle, windows):
  #Parámetros para la matriz de GLCM
  distances = [distance]  #Distancia entre los píxeles vecinos
  angles = [angle]  #Dirección del desplazamiento
  levels = 256  #Número de niveles de gris

  #Vectores de características de las ventanas
  features = []


  #Función que calcula la matriz de GLCM para cada ventana
  for window in windows:
    glcm_matrix = graycomatrix(window, distances=distances, angles=angles, levels=levels)

    #Características de Haralick de segundo orden
    contrast = graycoprops(glcm_matrix, 'contrast')
    homogeneity = graycoprops(glcm_matrix, 'homogeneity')
    energy = graycoprops(glcm_matrix, 'energy')
    correlation = graycoprops(glcm_matrix, 'correlation')
    mean = np.mean(glcm_matrix)
    entropy = -np.sum(glcm_matrix * np.log2(glcm_matrix + 1e-8))

    feature = []
    feature.append(contrast[0][0])
    feature.append(correlation[0][0])
    feature.append(homogeneity[0][0])
    feature.append(energy[0][0])
```

```
    feature.append(mean)
    feature.append(entropy)

    features.append(feature)
  df = pd.DataFrame(features, columns=['contrast', 'correlation', 'homogeneity','energy','mean','entropy'])
  return df
```

V-D. Función para la obtención del vector de características Haralick de las ventanas de las imágenes

```
def features_GLCM(nombre, tamanio, desplazamiento):
  #Divide la imagen en ventanas de tamaño y desplazamiento determinado por el usuario
  windows = divide_windows("Brodatz_examples/" + nombre, tamanio, desplazamiento)

  #Muestra la primer ventana generada
  plt.imshow(windows[0], cmap='gray')
  plt.axis('off')
  plt.show()

  #Distancia de 1, ángulo 0°
  features_df1_01 = get_GLCM(1,0,windows)
  features_df1_01['texture'] = nombre

  ##Distancia de 2, ángulo 90°
  features_df1_290 = get_GLCM(2,90,windows)
  features_df1_290['texture'] = nombre

  #print(features_df1_01)
  #print(features_df1_290)
  return features_df1_01,features_df1_290
```

V-E. Uso de funciones para la obtención de los vectores de características de las ventanas pertenecientes a las imágenes utilizadas para la creación de los modelos de caracterización

```
features_D16_01,features_D16_290 = features_GLCM("D16.bmp", 51, 5)
features_D64_01,features_D64_290 = features_GLCM("D64.bmp", 51, 5)
features_D46_01,features_D46_290 = features_GLCM("D46.bmp", 51, 5)
features_D101_01,features_D101_290 = features_GLCM("D101.bmp", 51, 5)
features_D6_01,features_D6_290 = features_GLCM("D6.bmp", 51, 5)
features_D49_01,features_D49_290 = features_GLCM("D49.bmp", 51, 5)
features_piedras_01,features_piedras_290 = features_GLCM("Piedras.jpg", 51, 5)

#Unión de DataFrames con vectores de características de las ventanas pertenecientes a las imágenes de
↪  texturas utilizadas.
df_final_01 = pd.concat([features_D16_01, features_D64_01, features_D46_01, features_D101_01,
↪  features_D6_01, features_D49_01, features_piedras_01], axis=0)
df_final_290 = pd.concat([features_D16_290, features_D64_290, features_D46_290, features_D101_290,
↪  features_D6_290, features_D49_290, features_piedras_290], axis=0)
```

V-F. Función para la implementación del Clasificador Bayesiano

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix

def Naive_Bayes(df_final):
  #Divide el DataFrame en variables (X) y etiqueta (y)
  X = df_final.iloc[:, :-1]
  y = df_final.iloc[:, -1]

  #Divide los datos en conjuntos de entrenamiento (80%) y prueba (20%)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

  #Crea un objeto de clasificador Bayesiano
  nb = GaussianNB()

  #Entrena el clasificador con los datos de entrenamiento
  nb.fit(X_train, y_train)

  #Realiza predicciones en el conjunto de prueba
  y_pred = nb.predict(X_test)

  #Calculamos la exactitud del modelo
```

```
  score = nb.score(X_test, y_test)
  print('Score:', score)

  #Calculamos la matriz de confusión del modelo
  conf_matrix = confusion_matrix(y_test, y_pred)
  print('Matriz de confusión:\n', conf_matrix)
  return nb
```

V-G. Función para la implementación del Clasificador KNN

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

def KNN(df_final):
  #Divide el DataFrame en variables (X) y etiqueta (y)
  X = df_final.iloc[:, :-1]
  y = df_final.iloc[:, -1]

  #Divide los datos en conjuntos de entrenamiento (80%) y prueba (20%)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

  #Crea un objeto de clasificador KNN con k=3
  knn = KNeighborsClassifier(n_neighbors=3)

  #Entrenamos el clasificador con los datos de entrenamiento
  knn.fit(X_train, y_train)

  #Realizamos predicciones en el conjunto de prueba
  y_pred = knn.predict(X_test)

  #Calculamos la precisión del modelo
  score = knn.score(X_test, y_test)
  print('Puntaje:', score)

  #Calculamos la matriz de confusión del modelo
  conf_matrix = confusion_matrix(y_test, y_pred)
  print('Matriz de confusión:\n', conf_matrix)
  return knn
```

V-H. Entrenamiento de los cuatro modelos de clasificadores

```
nb_01 = Naive_Bayes(df_final_01)
nb_290 = Naive_Bayes(df_final_290)
knn_01 = KNN(df_final_01)
knn_290 = KNN(df_final_290)
```

V-I. Función para la caracterización de texturas de nuevas imágenes

```
def Predecir(nombre, tamanio, desplazamiento, metodo):
  #Dividir la imagen en ventanas de 31x31 pixel con un desplazamiento de 10 en 10, tardo 8 min, funciona
  ↪  bien 31 y 10
  windows = divide_windows("texturas/" + nombre, tamanio, desplazamiento)

  # Mostrar la ventanas 1
  plt.imshow(windows[0], cmap='gray')
  plt.axis('off')
  plt.show()

  #distancia de 1, angulo 0
  features_df_compuesta_01 = get_GLCM(1,0,windows)
  if metodo == 1:
    y_predict = nb_01.predict(features_df_compuesta_01)
  elif metodo == 2:
    y_predict = nb_290.predict(features_df_compuesta_01)
  elif metodo == 3:
    y_predict = knn_01.predict(features_df_compuesta_01)
  elif metodo == 4:
    y_predict = knn_290.predict(features_df_compuesta_01)
  else:
    print("Error método incorrecto")
  y_predict
  features_df_compuesta_01['clase'] = y_predict
```

```
  clases = features_df_compuesta_01['clase'].value_counts()
  print(clases)
  return features_df_compuesta_01
```

V-J. Función para mostrar la caracterización obtenida a partir de los modelos de clasificación

```
from math import ceil, sqrt

def mostrar_Clasificacion(features_df_compuesta_01):
  # Obtener el número de filas del DataFrame
  n_filas = len(features_df_compuesta_01)

  # Calcular el tamaño de la imagen cuadrada
  tamaño = ceil(sqrt(n_filas))

  # Crear un arreglo vacío de tamaño tamaño x tamaño x 3 (ancho x alto x canales RGB)
  imagen = np.zeros((tamaño, tamaño, 3), dtype=np.uint8)

  # Recorrer por la última columna del DataFrame
  for i, etiqueta in enumerate(features_df_compuesta_01['clase']):
      # Si la etiqueta es "D10", agregar un píxel rojo en la posición (i, 0)
      if etiqueta == "D101.bmp":
          imagen[i // tamaño, i % tamaño] = [255, 0, 0]  # Rojo
      # Si la etiqueta es "D11", agregar un píxel verde en la posición (i, 0)
      elif etiqueta == "D64.bmp":
          imagen[i // tamaño, i % tamaño] = [0, 255, 0]  # Verde
      # Si la etiqueta es "D12", agregar un píxel amarillo en la posición (i, 0)
      elif etiqueta == "D46.bmp":
          imagen[i // tamaño, i % tamaño] = [255, 255, 0]  # Amarillo
      # Si la etiqueta es "D12", agregar un píxel amarillo en la posición (i, 0)
      elif etiqueta == "D16.bmp":
          imagen[i // tamaño, i % tamaño] = [0, 0, 255]  # azul
      elif etiqueta == "D6.bmp":
          imagen[i // tamaño, i % tamaño] = [160, 160, 160]  # gris
      elif etiqueta == "D49.bmp":
          imagen[i // tamaño, i % tamaño] = [255, 102, 255]  # Rosa
      elif etiqueta == "Piedras.jpg":
          imagen[i // tamaño, i % tamaño] = [178, 102, 255]  # Morado
  plt.imshow(imagen)
  plt.show()
```

V-K. Función para calcular la exactitud real de nuestros modelos en imágenes compuestas

```
import math
#print(imagen1)
#print(len(imagen1))
def get_color(imagen_original):
  if imagen_original == 'imgCompuesta1.png':
    return [[160, 160, 160],[0, 255, 0],[255, 102, 255],[255, 0, 0]]
  elif imagen_original == 'imgCompuesta2.png':
    return [[0, 0, 255],[0, 255, 0],[255, 255, 0],[255, 0, 0]]
  else:
   return [[255, 0, 0],[0, 255, 0],[255, 255, 0],[178, 102, 255]]
def generar_arreglo_pixeles(cantidad_pixeles, imagen_original): #cantidad de pixeles es len(imagen1)

    limite = cantidad_pixeles // 2

    arreglo_pixeles = []

    for i in range(cantidad_pixeles):
        fila = []
        for j in range(cantidad_pixeles):
            if i < limite:
              if j < limite:
                fila.append(get_color(imagen_original)[0])
              else:
                fila.append(get_color(imagen_original)[1])
            else:
              if j < limite:
                fila.append(get_color(imagen_original)[2])
              else:
                fila.append(get_color(imagen_original)[3])
        arreglo_pixeles.append(fila)
```

```
    return arreglo_pixeles
def get_iguales(imagen1, imagen2):
  iguales = 0
  dimension = len(imagen1)
  for i in range(dimension):
    for j in range(dimension):
      if np.array_equal(imagen1[i][j], imagen2[i][j]):
        iguales += 1

  return "La exactitud es de: "+str(iguales / (dimension * dimension))

def get_exactitud(imagen1, original):
 imagen2 = generar_arreglo_pixeles(len(imagen1), original)
 return get_iguales(imagen1, imagen2)
```

V-L. Caracterización de Imágenes Compuestas

```
features_Compuesta_01 = Predecir("imgCompuesta1.png", 51,5,1) #Utilizando el metodo 1
mostrar_Clasificacion(features_Compuesta_01)
get_exactitud(imagen1,'imgCompuesta1.png' )
features_Compuesta_02 = Predecir("imgCompuesta1.png", 51,5,2) #Utilizando el metodo 2
mostrar_Clasificacion(features_Compuesta_02)
get_exactitud(imagen1,'imgCompuesta1.png' )
features_Compuesta_03 = Predecir("imgCompuesta1.png", 51,5,3) #Utilizando el metodo 3
mostrar_Clasificacion(features_Compuesta_03)
get_exactitud(imagen1,'imgCompuesta1.png' )
features_Compuesta_04 = Predecir("imgCompuesta1.png", 51,5,4) #Utilizando el metodo 4
mostrar_Clasificacion(features_Compuesta_04)
get_exactitud(imagen1,'imgCompuesta1.png' )
features_Compuesta_05 = Predecir("imgCompuesta2.png", 51,5,3) #Utilizando el metodo 3, (KNN, 0 grados 1
↪  paso)
mostrar_Clasificacion(features_Compuesta_05)
get_exactitud(imagen1,'imgCompuesta2.png')
features_Compuesta_06 = Predecir("imgCompuesta3.png", 51,5,3) #Utilizando el metodo 3, (KNN, 0 grados 1
↪  paso)
mostrar_Clasificacion(features_Compuesta_06)
get_exactitud(imagen1,'imgCompuesta3.png' )
```

VI. CONCLUSIONES

A lo largo de esta práctica, se tuvo como principal objetivo el que el alumno lograra desarrollar métodos de caracterización de texturas y aprendiera a utilizar clasificadores como el Clasificador Bayesiano, el Clasificador K-NN o alguno otro. En este sentido, consideramos que los objetivos se cumplieron debido a que logramos desarrollar una práctica en la cual pudiéramos realizar la clasificación de texturas a través de imágenes compuestas, donde se mostraran las clases a las que fuera clasificado cada una de los pixeles de la imagen compuesta.

Del mismo modo, consideramos que la práctica fue muy útil para comprender de mejor manera el funcionamiento de los clasificadores Bayesianos, siendo una práctica complementaria a la práctica anterior, donde al trabajar tanto con Clasificadores Bayesianos como K-NN logramos obtener resultados bastante buenos para la caracterización de nuestras imágenes compuestas, siendo que pudimos haber obtenido resultados mejores y más completos si el poder de cómputo requerido no fuera tan exigente, siendo que decidimos optar por parámetros que nos permitieran obtener resultados muy buenos sin que el tiempo de división de ventanas, entrenamiento y prueba fueran excesivos, debido a que deseábamos a su vez presentar los resultados obtenidos durante la clase.

Con respecto a la caracterización de texturas, dentro de las imágenes compuestas logramos apreciar que se puede llegar a sufrir errores al momento de trabajar con desplazamientos muy grandes, siendo que existen problemas de clasificación y caracterización al momento de trabajar con áreas o ventanas cuyas regiones se encuentran compuestas por más de una textura.

Por otro lado, la práctica sirvió como un gran ejercicio sobre manejo de imágenes y texturas en Python, siendo que aplicamos diversos aprendizajes y conocimientos obtenidos de las prácticas previas como lo serían los métodos para la carga y preprocesamiento de imñnagenes así como el uso de Clasificadores como el Bayesiano, siendo que a partir de los Clasificadores Bayesianos y K-NN logramos predecir la pertenencia de cada uno de los pixeles de nuestra imagen con respecto a alguna de las siete clases de texturas con las que trabajábamos. Creemos que una de las principales dificultades a la que nos enfrentamos durante el desarrollo de la práctica fue la selección de parámetros para la obtención de los mejores resultados con respecto a la caracterización de texturas, debido a que a mayor tamaño de ventana y valor de desplazamiento, peor era el resultado obtenido, pero a menor tamaño de ventana y valor de desplazamiento el tiempo de ejecución y la cantidad de recursos necesarios para la ejecución era mucho mayor, por lo que se decidió trabajar con valores aceptables tanto en tiempo de ejecución, uso de recursos y eficacia de caracterización para nuestra implementación y análisis de resultados.

Como se mencionó previamente y para concluir con esta práctica queremos reafirmar el hecho de que consideramos que el objetivo de la práctica se cumplió en su totalidad, siendo que logramos obtener diversos métodos para la caracterización de texturas mediante el Clasificador Bayesiano y el K-NN. Con respecto a las métricas, pudimos apreciar grados de precisión altos al momento de generar nuestros modelos de caracterización, contando con una precisión del 97.08 % y 99.39 % para los Clasificadores Bayesianos así como una precisión del 98.93 % y 99.33 % para los Clasificadores K-NN. Sin embargo, al momento de trabajar con imágenes compuestas y empleando el modelo de Clasificación K-NN con distancia entre pixeles vecinos de uno y ángulo de 0º se obtuvo una precisión del 83.43 %, 85.59 % y 86.04 % para la caracterización de las texturas de nuestras tres imágenes compuestas, siendo esta una precisión real aplicada bastante aceptable en contraste al bajo tiempo de ejecución que se requirió para correr el código y realizar la clasificación. En contraste, al trabajar con parámetros distintos como lo fue un desplazamiento de dos pixeles y un ángulo de 90º, se obtuvo una precisión considerablemente baja, siendo esta del 21.91 % y 44.54 %, con lo cual podemos concluir que la importancia de los modelos no únicamente recae en los clasificadores a utilizar, sino también en las imágenes de textura y en los parámetros de nuestros algoritmos.

REFERENCIAS

[1] H.-B. Mryka, "Glcm texture: A tutorial. https://prism.ucalgary.ca/items/8833a1fc-5efb-4b9b-93a6-ac4ff268091c," 2017.

[2] IBM, "What are naive bayes classifiers?. https://www.ibm.com/topics/naive-bayes," 2023.

[3] Scikit-Learn, "Naive bayes. https://scikit-learn.org/stable/modules/naive$_b ayes.html$,'' 2023.

[4] E. A. F. C. R. O. Rodríguez Rodríguez, Jorge Enrique; Rojas Blanco, "Clasificación de datos usando el método k-nn. https://geox.udistrital.edu.co/index.php/vinculos/article/view/4111/5778," 2007.

[5] O. M. Jimena, "Bayesian inference. laboratorio avanzado de procesamiento de imágenes," 2022.