Práctica 4 “Caracterización y clasificacion de Texturas”

Leyva P. José Luis , *Facultad de Ingeniería, UNAM,* Lopez Cruz Luis Enrique, *Facultad de Ingenieria, UNAM*
Najera S. Isaac Israel , *Facultad de Ingeniería, UNAM,* Sandoval M. Joaquin, *Facultad de Ingeniería, UNAM,*

***Resumen*—En esta práctica se intenta reconocer y clasificar texturas basándose en la matriz GLCM de cada textura y sus estadísticos de segundo orden. Se utilizan distintos clasificadores, como el clasificador Bayesiano, el clasificador de K-vecinos más cercanos y la máquina de soporte vectorial. Posteriormente, se evalúan los clasificadores intentando reconocer la textura y coloreándola de acuerdo a su clasificación. Los resultados, aunque no excelentes, son buenos y nos permiten entender más cómo utilizar las texturas para el análisis de imágenes.**

***Index Terms*—IEEE, IEEEtran, journal, LaTeX, paper, template.**

I. Objetivo

El alumno desarrollará métodos de caracterización de texturas y aprendera a utilizar clasificadores como K-NN, K-MEANS o maquina de soporto vectorial

II. Introducción

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 irregular en función de la variación espacial de las intensidades de píxeles. En ese sentido, la rugosidad o la irregularidad se refieren a las variaciones de los valores de intensidad o niveles de gris.

El análisis de texturas se utiliza en varias aplicaciones, como la detección remota, la inspección automatizada y el procesamiento de imágenes médicas. El análisis de texturas puede utilizarse para identificar los límites de las regiones que comparten una textura común, lo que se denomina segmentación de texturas. El análisis de texturas puede ser útil cuando los objetos en una imagen se caracterizan más por su textura que por su intensidad y las técnicas de umbral tradicionales no son efectivas.

Históricamente, el método más común para describir la información de la textura es el enfoque estadístico, el cual incluye los métodos estadísticos de primer orden, segundo orden y órdenes más altos. Estos métodos analizan la distribución de propiedades específicas de la imagen usando el valor de sus pixeles. Particularmente, el método de segundo orden es importante ya que toma en cuenta la distribución de las intensidades de los pixeles y además su posición espacial sin sacrificar mucho tiempo en los cálculos, siendo el caso de los métodos de órdenes más altos.

II-A. Algoritmos de clasificacion

II-A1. K-NN: KNN es un algoritmo de aprendizaje supervisado, que a partir de un conjunto de datos iniciales clasificara todas las instancias nuevas en el grupo corresponda, esto se hace en base a calcular la distancia del elemento a clasificar con cada uno de los existentes, ordenando estas distancias de menor a mayor para ir seleccionando al grupo que pertenece, el grupo se seleccionara con la mayor frecuencia y menor distancia. Antes de que se pueda hacer una clasificación, se define la distancia. La distancia usualmente más utilizada es la euclidiana.

El valor de k en el algoritmo define a los vecinos que se verificarán para determinar la clasificación de un punto de consulta específico. Por ejemplo, si k=1, la instancia se asignará a la misma clase que su vecino más cercano. La elección de k dependerá en gran medida de los datos de entrada, suele ser una tarea que se hace mayormente con equilibrio para evitar un ajuste excesivo o incluso hasta inexacto

II-A2. K-Means: K-means es un método de aprendizaje no supervisado para agrupar puntos de datos. El algoritmo divide iterativamente los puntos de datos en K clústeres minimizando la varianza en cada clúster.

II-A3. Maquina de soporte vectorial (Svm): Es un algoritmo de aprendizaje automático supervisado que se puede utilizar para problemas de clasificación o regresión. Generalmente se usa para clasificar. Dada algunas clases de datos etiquetadas, este actuara como un clasificador discriminativo, definido formalmente por un hiperplano óptimo que separa todas las clases.

III. Desarrollo

Lo primero que se realizo fue la creacion funciones junto con la eleccion de un tamaño adecuado de las ventanas para la obtencion de los texeles.

```
def getTexelData(image_path: str):
  textura = plt.imread(IMAGE_PATH+image_path)
  textura = cv2.cvtColor(textura, cv2.COLOR_RGB2GRAY
    )

  ubicacion_texels = [(0,0),(200,150),(250,350)
    ,(100,100),(50,400),(400,100),(300,120),
                      (123,342),(421,111),(23,33)
    ,(265,31),(78,473),(300,300),(400,212)]
  texels = getPatches(textura, ubicacion_texels,40)
  return getTrainingData(texels)
```

III-A. GLCM

Al tener definidos los texels de cada imagen, obtuvimos la información característica para cada uno de ellos, con base en el proceso de extracción de características o GLCM o gray level coocurrence matrix. Afortunadamente la biblioteca skimage incluye una funcionalidad que le viene como anillo al dedo a nuestro propósito ya que permite variar parámetros específicos como el ángulo y la distancia. El siguiente código muestra brevemente como la utilizamos.

Una vez determinada nuestra matriz GLCM, el paso siguiente fue obtener los estadisticos de 2do grado. Con la función ”graycorps”de la libreria skimage calculamos las propiedades de la matriz GLCM, en concreto se podia especificar los atributor a calcular como: Contraste, homogeneidad, ASM energia, correlación, etc.

```
from skimage.feature import graycomatrix,
    graycoprops
def getFeatureVector(texel: np.ndarray, stats: list)
    :

    glcm = graycomatrix(texel,distances=[5],angles
    =[0],levels=256,symmetric=True, normed=True)
    vect = []

    for stat in stats:
        vect.append(graycoprops(glcm,prop=stat)
    [0,0])
    return vect
```

III-B. clasificacion con Naive Bayes

Para poder desarrollar un clasificador basado en Naive Bayes y sus subsecuentes nos apoyamos de la biblioteca scikit-learn, la cual ya habíamos utilizado en la práctica pasada. De acuerdo con nuestra experiencia pasada utilizamos la versión “GaussianNB” que cuyo nombre indica aplica un filtrado gaussiano. El código base para utilizar dicha biblioteca es el siguiente:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(tr_data,tr_lbls)
```

En este caso, obtenemos los datos de entrenamiento y las etiquetas con el siguiente código y las siguientes funciones, que iremos explicando:

```
images  = ['D51.bmp','D85.bmp','D6.bmp','D20.bmp','
    D103.bmp','D64.bmp','D52.bmp','D17.bmp']
tr_data = []
tr_lbls = []
for img in images:
  tr_data += getTexelData(img)
tr_lbls = [i//(len(tr_data)//8) for i in range(len(
    tr_data))]
```

En el código anterior, iteramos sobre las imágenes de entrenamiento para obtener los datos de entrenamiento, que se almacenan en `tr data`. Como es necesario mandar etiquetas para los clasificadores, creamos una lista donde obtenemos su etiqueta de acuerdo al número de elementos e imágenes.

III-C. clasificacion con K-NN

De manera homologa al clasificador de Naive Bayes, la implementación de un clasificador KNN fue sencilla debido a la existencia de la función en la libreria scikit-learn y una vez teniendo nuestras clases definidas pudimos entrenar el algotimo. Nuestro valor k fue determinado por un equilibrio ya que probamos varios y nos quedamos con el que vimos con un mejor resultado e investigando nos dimos cuenta que es mejor el algoritmo con un valor k inpar y que los valores más bajos de k pueden tener una varianza alta, pero un sesgo bajo, y los valores más grandes de k pueden generar un sesgo alto y una varianza más baja

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(tr_data,tr_lbls)
```

III-D. SVC

Similar a los otros 2 algoritmos de clasificación utilizamos una función de la libreria sklearn con la que implementamos este algoritmo con las caracteristicas de nuestros datos y las etiquetas de nuestros grupos donde clasificaremos los nuevos datos. Este algoritmo devuelve un hiperplano que divide y categoriza nuestros datos.

```

svm = SVC()
svm.fit(tr_data,tr_lbls)
```

Una vez que se termino el entrenamiento de nuestras 8 imágenes y se evaluaron las características de su texels, se creo una imagen compuesta por cuatro de las texturas elegidas para el entrenamiento, pero procurando utilizar regiones diferentes.

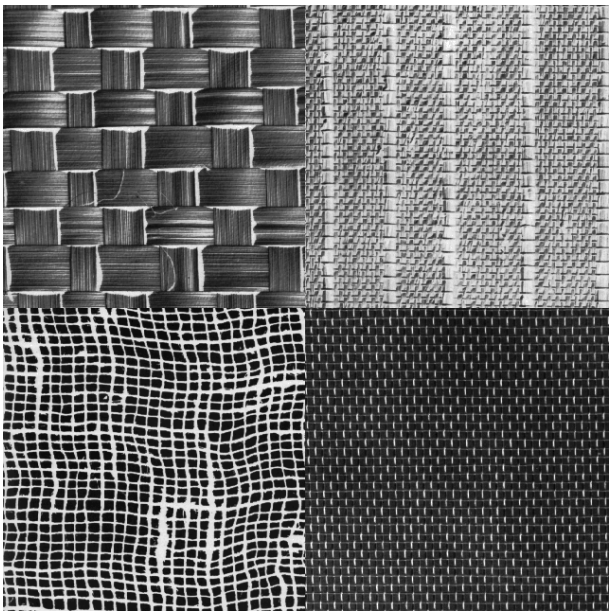
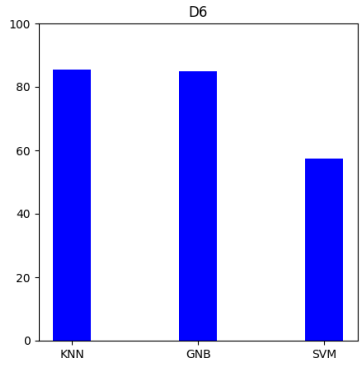


Figura 1: Imagen compuesta de texturas

IV. RESULTADOS

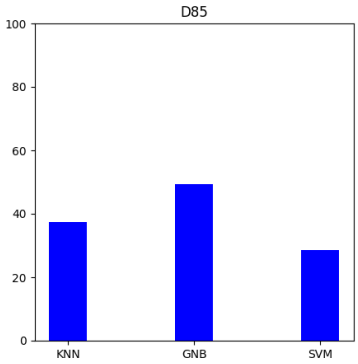
Después de evaluar y clasificar cada una de las texturas utilizadas pudimos observar nuevamente y comparar los distintos resultados obtenidos por los clasificadores, para nuestra sorpresa Naive Bayes y KNN fueron los mejores en clasificar

la mayoría de las imágenes a diferencia de SMV que solo destaca en la clasificación de dos texturas. Sin embargo, creemos que si se incremente en mayor medida el número de texturas y texeles los resultados podrían llegar a variar. Pera visualizar de mejor manera estos resultados obtuvimos las siguientes graficas :



D6

(a) D6



D85

(b) D85

Figura 2: graficas

IV-A. Variación parámetros Matriz GLCM

La función `graycomatrix()` tiene como parámetros, además de la imagen a evaluar, dos parámetros que podemos variar para ver su efecto. Estos son distancia entre el offset de pixeles, y el ángulo del análisis en radianes.

Todos los cálculos los realizamos con una distancia de 5 píxeles y 0 grados de inclinación. Para la prueba utilizamos una distancia de 10 pixeles y 90 grados de inclinación. La imagen sobre la cual hicimos las pruebas fue la `D103.bmp`:

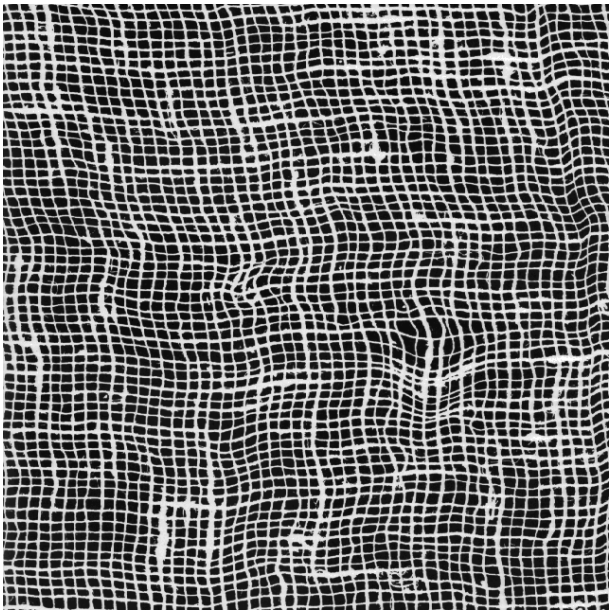


Figura 3: D103.bmp

La imagen con los parámetros originales es clasificada de la siguiente manera:

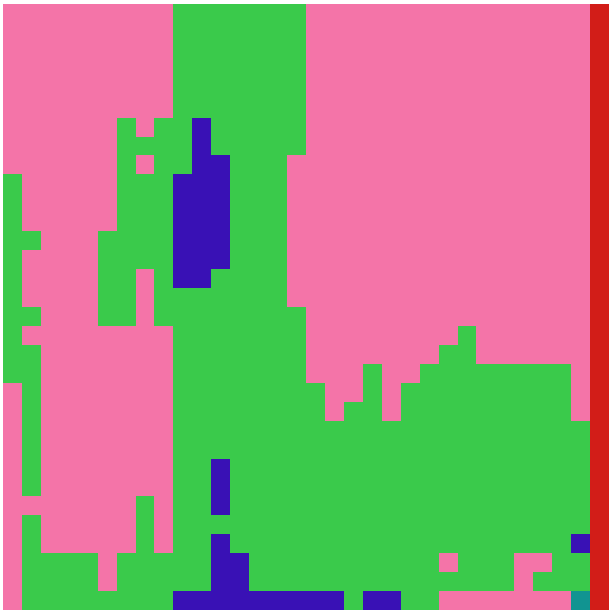


Figura 4: D103 parámetros originales

La imagen con los nuevos parámetros de la matriz GLCM queda de la siguiente manera:

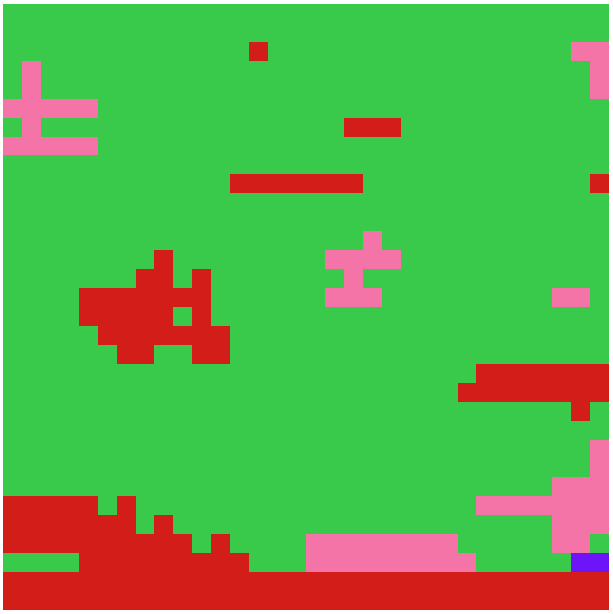


Figura 5: D103 con parámetros modificados

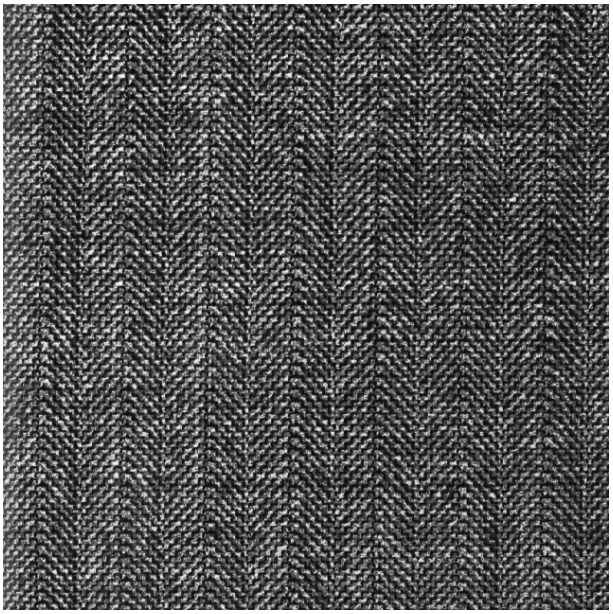


Figura 6: D17.bmp

Como se puede observar, los resultados varían bastante, aunque la clasificación sigue siendo acertada. Parece ser que la variación del ángulo ayuda a mejorar la clasificación; puede que se deba a que la textura original también tiene una cierta inclinación. Con esto comprobamos que la variación mínima de la matriz GLCM puede tener un gran impacto en cómo reacciona nuestro clasificador.

IV-B. Variación de Estadísticos

Para evaluar el efecto que tenían los estadísticos en los resultados de la clasificación, utilizamos la función `getFeatureVector()` pero le mandamos una lista de estadísticas distinta. Posteriormente, volvimos a obtener los datos de prueba y entrenamos los clasificadores, para después evaluarlos y comparar las diferencias. Realizamos 2 pruebas y obtuvimos los siguientes resultados.

Las pruebas las realizamos con la imagen `D17.bmp`, donde el color correcto de clasificación era el amarillo. En general, nuestros resultados fueron casi siempre acertados, con muy pocas variaciones, como se verá a continuación.

La imagen original es la siguiente (figura 6).

Los resultados de clasificación usando todos los estadísticos son los siguientes (figuras 7, 8, 9):

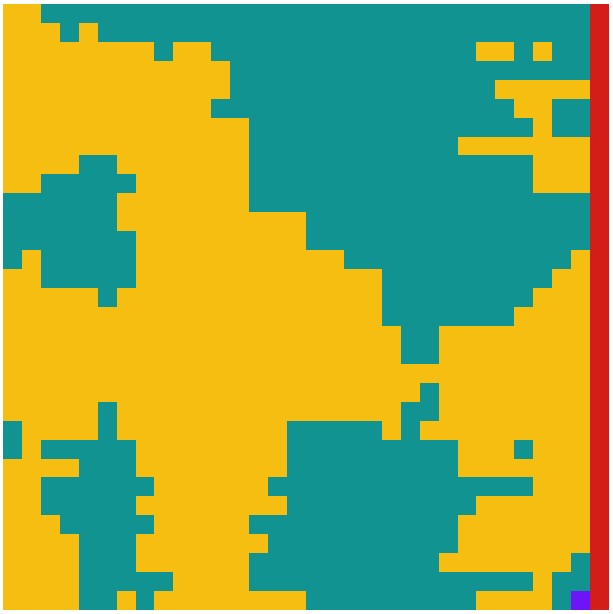


Figura 7: Gaussian Naive Bayes y Todos los Estadísticos

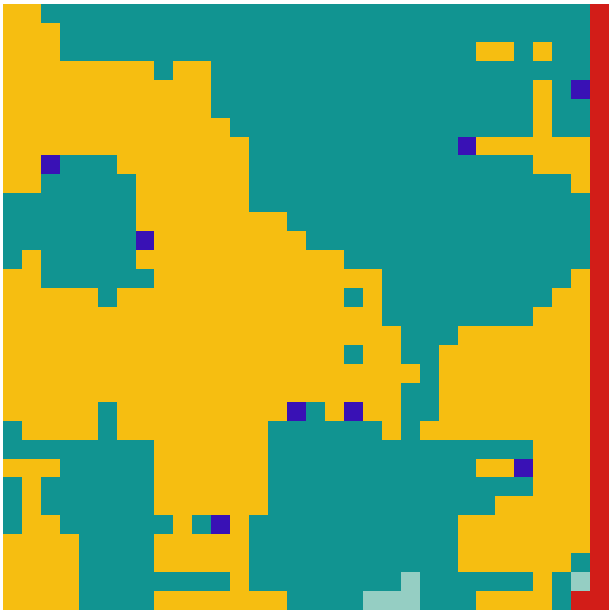


Figura 8: Máquina de Soporte Vectorial y Todos los Estadísticos

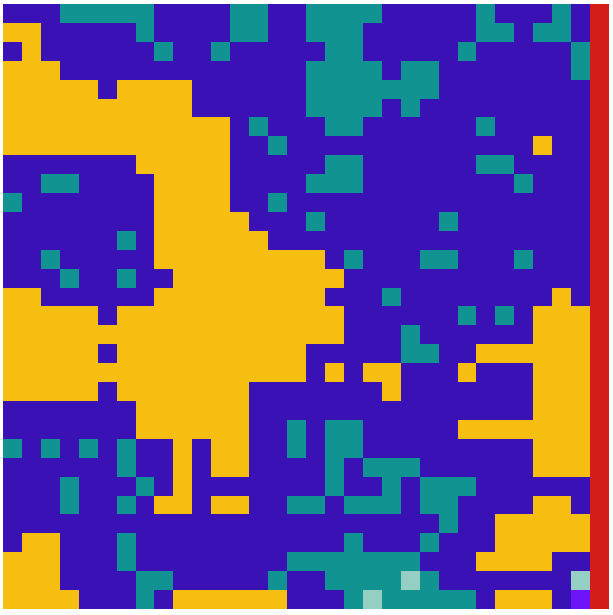


Figura 9: K-nearest neighbour y Todos los Estadísticos

Como se puede ver, tanto la máquina de soporte vectorial como el clasificador Bayesiano tienen resultados muy similares y en su mayoría acertados, a diferencia del clasificador KNN.

Para nuestra primera prueba utilizamos los estadísticos de **disimilitud, homogeneidad, ASM y energía**. Los resultados son los siguientes (figuras 10, 11, 12) :

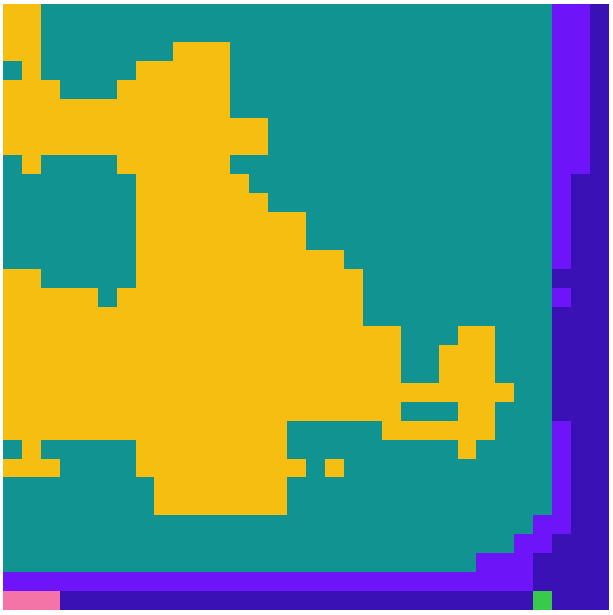


Figura 10: Gaussian Naive Bayes y prueba 1

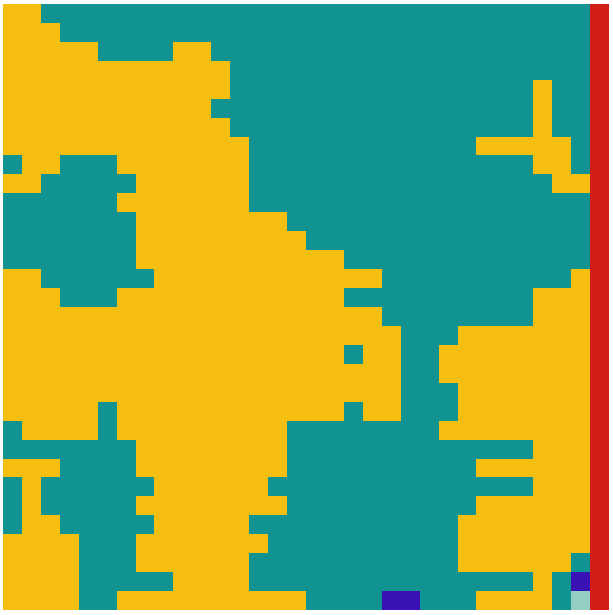


Figura 11: Máquina de Soporte Vectorial y prueba 1

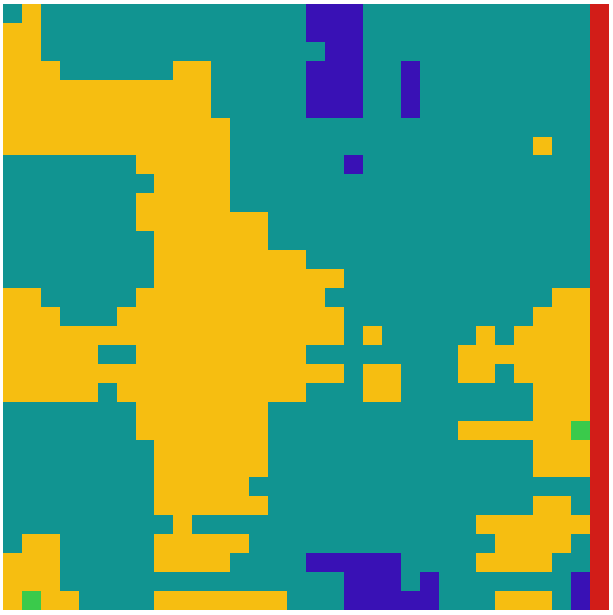


Figura 12: K-nearest neighbour y prueba 1

Como podemos ver, los resultados varían, siendo el cambio más notable la mejora en la clasificaicón del clasificador KNN. Sin embargo, ninguno llega a clasificar al 100 % la imagen.

En la siguiente prueba utilizamos los estadísticos de **contraste, disimilitud, energía y correlación.** Los resultados fueron los siguientes:

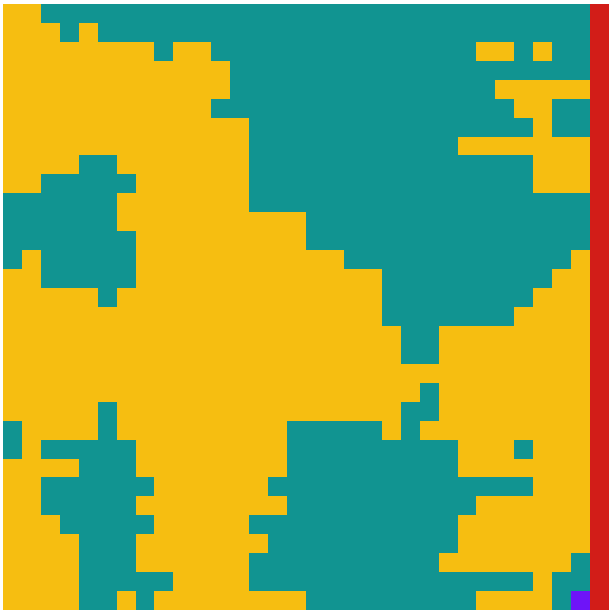


Figura 13: Gaussian Naive Bayes y prueba 2

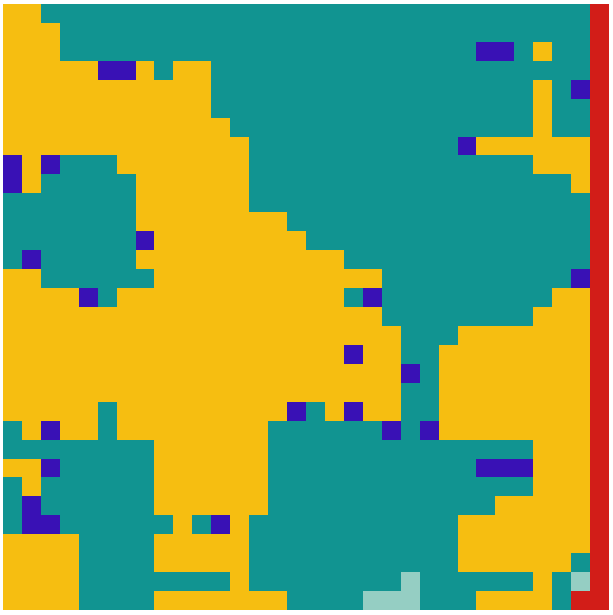


Figura 14: Máquina de Soporte Vectorial y prueba 2

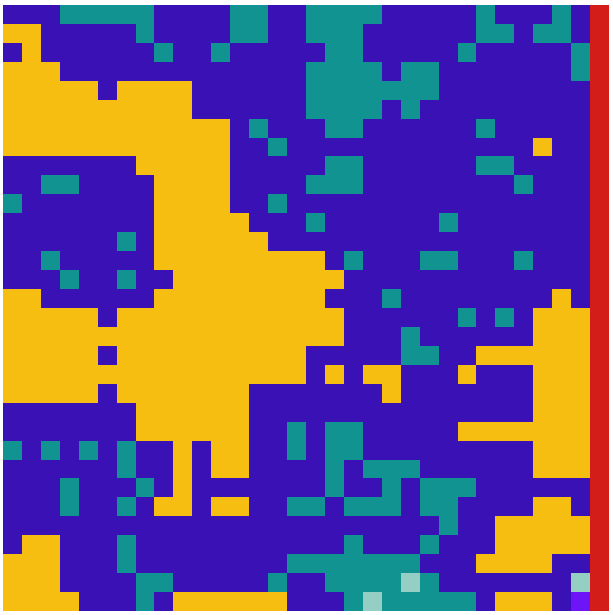


Figura 15: K-nearest neighbour y prueba 2

Podemos considerar esta prueba como fallida, pues volvimos a los resultados de usar todos los estadísticos. Con esta información, podemos intuir que el estadístico de contraste y correlación son los que más ruido.ºcasionan a la hora de entrenar el clasificador, pues cuando estos estadísticos no se consideraban, la clasificación mejoró un poco. Harían falta más pruebas, con mas variaciones de estadísicos y otras imágenes para poder llegara a una conclusión más certera.

IV-C. Textura Compuesta

El último ejercicio era la evaluación de nuestro clasificador en una imagen con varias texturas. Utilizamos el clasificador de bayes, pues fue el que mejor desempeño tenía. Además,

usamos todos los estadísticos y la siguiente imagen de texturas compuestas:

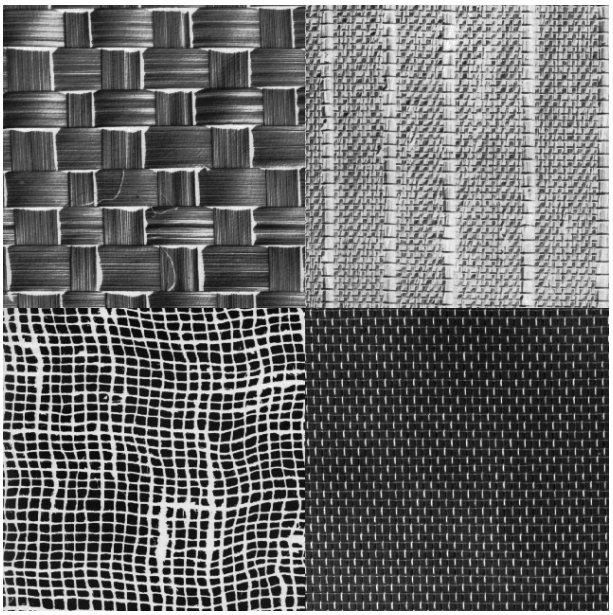


Figura 16: Imagen compuesta

Los resultados de la clasificación son los siguientes:

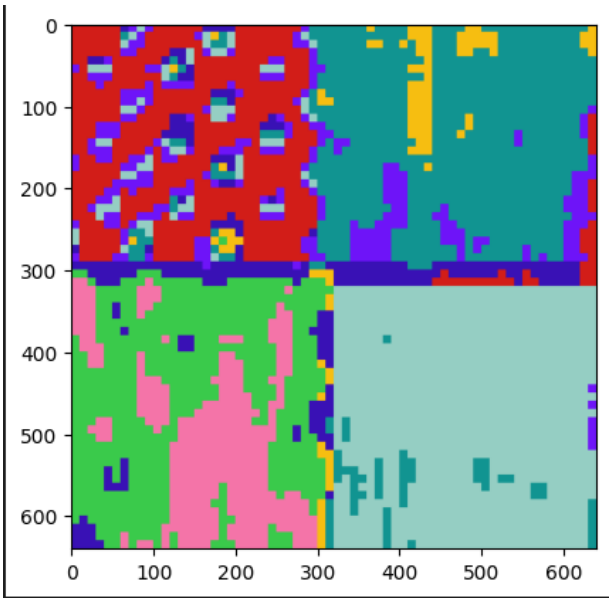


Figura 17: Evaluacion sobre imagen compuesta

A pesar de que la clasificación no es perfecta, en cada textura predomina la clasificación correcta. Además, se puede observar claramente donde empiezan y terminan las texturas; es decir, se observa la cuadrícula de la textura compuesta, por lo que consideramos que nuestro clasificador funciona correctamente.

V. CÓDIGO

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
from skimage.feature import graycomatrix, graycoprops
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from typing import Union

IMAGE_PATH = "./Brodatz_examples/"


def getPatches(image: np.ndarray, locations: list, patch_size=20) -> list:
    res = []
    for loc in locations:
        res.append(image[loc[0] : loc[0] + patch_size, loc[1] : loc[1] + patch_size])
    return res


def getFeatureVector(texel: np.ndarray):

    glcm = graycomatrix(
        texel, distances=[5], angles=[0], levels=256, symmetric=True, normed=True
    )
    return [
        graycoprops(glcm, prop="contrast")[0, 0],
        graycoprops(glcm, prop="dissimilarity")[0, 0],
        graycoprops(glcm, prop="homogeneity")[0, 0],
        graycoprops(glcm, prop="ASM")[0, 0],
        graycoprops(glcm, prop="energy")[0, 0],
        graycoprops(glcm, prop="correlation")[0, 0],
    ]


def getTrainingData(texels: list):
    vectors = []
    for tx in texels:
        vectors.append(getFeatureVector(tx))
    return vectors


def getTexelData(image_path: str):
    textura = plt.imread(IMAGE_PATH + image_path)
    textura = cv2.cvtColor(textura, cv2.COLOR_RGB2GRAY)

    ubicacion_texels = [
        (0, 0),
        (200, 150),
        (250, 350),
        (100, 100),
        (50, 400),
        (400, 100),
        (300, 120),
        (123, 342),
        (421, 111),
        (23, 33),
        (265, 31),
        (78, 473),
        (300, 300),
        (400, 212),
    ]
    texels = getPatches(textura, ubicacion_texels, 40)
    return getTrainingData(texels)


def sliding_window(image, stepSize, windowSize):
    for y in range(0, image.shape[0], stepSize):
        for x in range(0, image.shape[1], stepSize):
```

```
            yield (x, y, image[y : y + windowSize[1], x : x + windowSize[0]])


def evaluarClasificador(
    clasificador: Union[GaussianNB, SVC, KNeighborsClassifier],
    imagen: np.ndarray,
    windowStep: int,
    windowSize: tuple,
) -> np.ndarray:
    colores = [
        [57, 17, 181],
        [17, 148, 145],
        [149, 206, 195],
        [244, 116, 168],
        [58, 202, 75],
        [210, 29, 24],
        [110, 20, 248],
        [246, 190, 17],
    ]
    res = np.zeros(shape=imagen.shape)
    res = np.dstack([res, res, res])
    for (x, y, window) in sliding_window(imagen, windowStep, windowSize):

        vector = getFeatureVector(window)

        color_prediction = colores[int(clasificador.predict([vector]))]
        for iy in range(y, y + 10):
            for ix in range(x, x + 10):
                if ix < res.shape[1] and iy < res.shape[0]:
                    res[iy, ix] = color_prediction

    return res


images = [
    "D51.bmp",
    "D85.bmp",
    "D6.bmp",
    "D20.bmp",
    "D103.bmp",
    "D64.bmp",
    "D52.bmp",
    "D17.bmp",
]

lbl = ["KNN", "GNB", "SVM"]

colores = [
    [57, 17, 181],
    [17, 148, 145],
    [149, 206, 195],
    [244, 116, 168],
    [58, 202, 75],
    [210, 29, 24],
    [110, 20, 248],
    [246, 190, 17],
]


tr_data = []
tr_lbls = []
k = 0
for img in images:
    tr_data += getTexelData(img)
    tr_lbls += [k for _ in range(14)]
    k += 1

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(tr_data, tr_lbls)

gnb = GaussianNB()
gnb.fit(tr_data, tr_lbls)


svm = SVC()
svm.fit(tr_data, tr_lbls)

img1 = plt.imread("./Brodatz_examples/D6.bmp")
img1 = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
test1 = evaluarClasificador(knn, img1[480:, 480:], 5, (40, 40))
plt.imshow(test1.astype(int))

clasificadores = [knn, gnb, svm]


for key_img, img_path in enumerate(images):
    img = plt.imread("./Brodatz_examples/" + img_path)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    precision = [0, 0, 0]
    for key, clasificador in enumerate(clasificadores):
        test = evaluarClasificador(clasificador, img[480:, 480:], 5, (40, 40))
        test = test.astype(int)
        for ix in range(test.shape[0]):
            for iy in range(test.shape[1]):
                color = test[ix, iy]
                precision[key] += 1 if (color == colores[key_img]).all() else 0
    accuracy = np.array(precision) / (160 * 160) * 100
    name = img_path.split(".")[0] + "_resultados.png"
    fig = plt.figure(figsize=(5, 5))
    plt.bar(lbl, accuracy, color="blue", width=0.3)
    plt.ylim(0, 100)
    plt.title(img_path.split(".")[0])
    plt.savefig("./resultados/" + name)


compuesta = plt.imread("./imagenes_combinadas/textura_compuesta3.bmp")
compuesta = cv2.cvtColor(compuesta, cv2.COLOR_RGB2GRAY)

res = evaluarClasificador(gnb, compuesta, 10, (40, 40))
plt.imshow(res.astype(int))
plt.imsave("./resultados/compuesta3.png", res.astype(np.uint8))
```

VI. Conclusion

En esta practica se utilizaron conceptos importantes para el procesamiento de imagenes, siendo uno de estos el concepto de matriz GLCM, que pueden mejorar la discriminacion de las texturas, ya que permite extraer un buena cantidad de informacion gracias a las relaciones espaciales de píxeles de una imagen. Para obtener buenos resultados hay que realizar una buena eleccion en el tamaño de la ventana, ademas de elegir texeles que proporcionen informacion que permita caracterizar a la clase exitosamente.

Referencias

[1] Análisis de texturas - MATLAB Simulink - MathWorks América Latina. (s.f.). Recuperado 30 de octubre de 2022, de https://la.mathworks.com/help/images/texture-analysis-1.html

[2] MÉTODOS DE ANÁLISIS DE TEXTURA - VELOCIDAD DE DECANTACIÓN. (s.f.). Recuperado 30 de octubre de 2022, de https://1library.co/article/m%C3%A9todos-an%C3%A1lisis-textura-velocidad-decantaci%C3%B3n.yj75672y

[3] K-Nearest Neighbors Algorithm Using Python. (2020, 25 noviembre). Edureka. Recuperado 24 de octubre de 2022, de https://www.edureka.co/blog/k-nearest-neighbors-algorithm/

[4] 1.9.Naive Bayes. (s.f.). scikit-learn. Recuperado 24 de octubre de 2022, de https://scikit-learn.org/stable/modules/naive_bayes.html
[5] Maquina de Soporte Vectorial (SVM). (2021). Medium. Recuperado 31 de octubre de 2022, de https://medium.com/@csarchiquerodriguez/maquina-de-soporte-vectorial-svm-92e9f1b1b1ac
[6] GLCM Texture Features. (s. f.) scikit-learn. Recuperado 31 de octubre de 2022, de https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_glcm.html