Práctica 3: Caracterización y Clasificación de texturas

Serapio Hernández Alexis Arturo
Facultad de Ingeniería
UNAM
Ciudad de México, México
alexis.serapio@ingenieria.unam.

García López Erik
Facultad de Ingeniería
UNAM
Ciudad de México, México
erikpumas999@gmail.com

Reyes Herrera Rogelio
Facultad de Ingeniería
UNAM
Ciudad de México, México
masterfive5of@gmail.com

***Abstract*—This article explores texture characterization and classification using GLCM-based feature extraction and classifiers like KNN and SVM. Tests with the Brodatz dataset highlight the impact of spatial relationships on model accuracy, achieving up to 99% under optimal conditions.**

***Keywords*—Basic Images Processing, Python, Pattern, Pattern Recognition, KNN, Bayes, SVM, classifier, Brodartz, Texture analysis.**

I. Objetivos

- El alumno desarrollará métodos de caracterización de texturas.
- Aprenderá a utilizar clasificadores como K-NN, K-Means o máquinas de soporte vectorial.

II. Introducción

El análisis y la caracterización de texturas juegan un rol fundamental en el procesamiento y análisis de imágenes, con aplicaciones que van desde la clasificación de superficies en imágenes satelitales hasta la detección de patrones en tejidos médicos como hemos visto en clase.

La textura en este contexto se refiere a las variaciones espaciales en la intensidad o color de una imagen, que proporcionan información sobre la estructura de los objetos presentes en la misma.

Para abordar la tarea de caracterizar texturas, es esencial aplicar técnicas que capturen estas variaciones de manera cuantitativa. En este sentido, métodos como las matrices de concurrencia, el análisis de frecuencia (mediante la transformada de Fourier o el análisis de Gabor) y los histogramas de gradientes, permiten obtener descripciones robustas que sirven como base para la clasificación.

Una vez que se caracterizan las texturas, se procede a clasificarlas. Entre los métodos más comunes para este propósito se encuentran los clasificadores como K-Nearest Neighbors (KNN), K-Means y las máquinas de soporte vectorial (SVM). El clasificador es un enfoque basado en instancias, que asigna las clases de una muestra en función de las etiquetas de sus vecinos más cercanos.

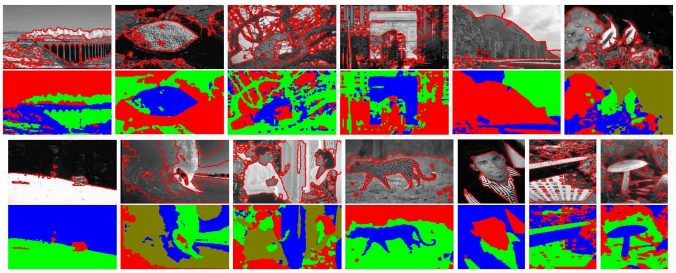
- **K-NN:** Es un enfoque basado en instancias que asigna la clase de una muestra en función de las etiquetas de sus vecinos más cercanos.

- **K-Means:** Es un algoritmo no supervisado que agrupa muestras en base a la similitud de sus características.

- **SVM:** Son técnicas que buscan maximizar el margen de separación entre clases mediante hiperplanos, logrando una clasificación robusta incluso en espacios de alta dimensionalidad.

De igual manera como realizamos la caracterización y clasificación, es necesario evaluar el rendimiento de este tipo de clasificadores ya que es necesario aplicar métodos de validación adecuados. Las técnicas más utilizadas incluyen la validación cruzada y la partición del conjunto de datos en subconjuntos de entrenamiento y prueba, lo que permite estimar de manera precisa la capacidad de generalización de los modelos.

Esta práctica entonces tiene como objetivos investigar y desarrollar técnicas de caracterización de texturas, aplicar los clasificadores mencionados y evaluar su rendimiento utilizando métricas estándar en el campo del aprendizaje automático.



III. DESARROLLO PRÁCTICA 1

3.1.- 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

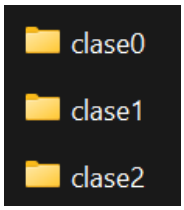
Usando el archivo zip llamado Brodatz, seleccionamos 4 imágenes de ladrillos, 3 de palma y 3 de círculos para el sistema.

Así se ve en el directorio de la computadora:

Clase 0: Ladrillos (4)

Clase 1: Palma (3)

Clase 2: Círculos (3)



3.2.- Usar de 8 a 12 imágenes texturas de la base de datos Brodatz para el proceso.

En total tenemos 10 imágenes las cuales tienen un preetiquetado dependiendo el directorio en el que se encuentren, es decir, tenemos una carpeta llamada clase 0, clase 1 y clase 2 para que, al cargarlas en Python, mientras trabaje con esos archivos ya se sepa a qué clase pertenece y agregarlo a una lista de etiquetas.

```
if os.path.isfile(file_path):
    imagen = io.imread(file_path)
    imagenes.append(imagen)
    etiquetas.append(clase) #Aqui se hace el etiquetado dependiendo de la carpeta origen
```

3.3.- De cada imagen tomar en "n" ventanas cuadradas (escoger las dimensiones adecuadas del texel de acuerdo a las texturas que utilice) y guardar de 2,3 o 4 de las mismas para el proceso de prueba siendo el resto para el proceso de entrenamiento.

Definimos la ventana de tamaño 64x64 ya que las imágenes son de 640x640, así generamos 100 vectores de características por imagen, suficientes para entrenar al modelo.

Estas ventanas se van guardando en una lista; en este paso optamos por el método tradicional y guardamos todo en la lista X para después dividir el "set de datos" en 80% entrenamiento y 20% pruebas.

```
for ventana in ventanas:
    caracteristicas = extraer_caracteristicas_glcm(ventana, distances, angles)
    X.append(caracteristicas)

X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split( *arrays: X, ventanas_e
                                                                         random_state=42)
```

3.4.- Obtener información característica de las imágenes a partir del proceso de extracción de características que entrega la matriz de Haralick o GLCM (gray level coocurrence matrix). Generar al menos 2 matrices de Haralick variando los parámetros de distancia y ángulo.

Usamos la biblioteca *from skimage.feature import graycomatrix, graycoprops* para hacer más fácil el proceso donde en una función aparte convertimos la imagen en escala de grises si es que no estaba así.

Graycomtatrix construye la matriz de Co-ocurrencia y recibe los siguientes parámetros:

- ventana_gris: La imagen en escala de grises
- distances: Lista de distancias entre píxeles que se comparan (por ejemplo, 1, 2, 3 píxeles de distancia).
- angles: Lista de ángulos de comparación (0°, 45°, 90°, 135°, en radianes).
- levels=256: Niveles de intensidad en la imagen (8 bits, 256 niveles de gris).
- symmetric=True: Hace que la matriz sea simétrica (GLCM cuenta ambas direcciones).
- normed=True: Normaliza los valores de la matriz para que sumen 1.

Entonces en la variable GLCM guardamos la matriz donde cada celda indica la frecuencia con la que una pareja de valores de gris (por ejemplo, [i, j]) aparece con la distancia y ángulo especificados

```
for dist in distances:
    for angle in angles:
        # Construir matriz de Co-ocurrencia (en niveles de gris)
        glcm = graycomatrix(ventana_gris.astype(np.uint8), distances=[dist], angles=[angle],
                            levels=256, symmetric=True, normed=True)
```

Con las configuraciones definidas en un diccionario recibidas como parámetro se calculaban los vectores de características tomando en cuenta las siguientes:

```
# Extraer caracteristicas de GLCM
contraste = graycoprops(glcm,  prop: 'contrast')[0, 0]
homogeneidad = graycoprops(glcm,  prop: 'homogeneity')[0, 0]
energia = graycoprops(glcm,  prop: 'energy')[0, 0]
entropia = -np.sum(glcm * np.log2(glcm + (glcm == 0)))  # Entropia
caracteristicas.extend([contraste, homogeneidad, energia, entropia])
```

- Contraste: Variación de intensidad entre un píxel y sus vecinos.
- Homogeneidad: Similitud entre los valores de la matriz.
- Energía: Uniformidad de la imagen (si es más regular o no).
- Entropía: Mide la cantidad de información contenida en la imagen (mayor desorden, mayor entropía).

Las configuraciones fueron muy variadas y en efecto tuvieron un impacto grande en el modelo de clasificación.

```python
configurations = [
    {'distancia': [1], 'orientacion': [0]},
    {'distancia': [3], 'orientacion': [0]},
    {'distancia': [1], 'orientacion': [np.pi / 4]},
    {'distancia': [1], 'orientacion': [3 * np.pi / 4]},
    {'distancia': [1, 2], 'orientacion': [0, np.pi / 4]},
    {'distancia': [1, 2], 'orientacion': [np.pi / 2, 3 * np.pi / 4]},
    {'distancia': [1, 3], 'orientacion': [0, np.pi / 4, np.pi / 2, 3 * np.pi / 4]},
    {'distancia': [1, 4], 'orientacion': [0, np.pi / 2]},
    {'distancia': [1, 2, 3], 'orientacion': [0, np.pi / 4, np.pi / 2]}
]
```

Se utilizaron combinaciones con las distancias 1,2,3 y 4; y con las direcciones 0° (horizontal), 45° (diagonal ascendente derecha), 90° (vertical), 135° (diagonal ascendente izquierda).

Las primeras suelen arrojar resultados con accuracy del 60%-82%, un poco deficiente, pero tomando en cuenta más orientaciones y distancias simultáneamente el modelo se hizo más robusto y alcanzamos hasta 99% de accuracy.

3.5.- De la matriz de Haralick obtener datos como los estadísticos de 2o grado, ej: media, entropía, energía u otra información; con estos datos generar un vector de características que serán los mismos tanto para el entrenamiento como para la prueba.

En la función de evaluación de clasificadores llamábamos a la función de extracción de características y tomando en cuenta cada una de las ventanas y las configuraciones para así agregarlos al vector general X y dividirlo como se mencionó previamente.

```python
def evaluar_clasificadores(distances, angles):
    X = []

    for ventana in ventanas:
        caracteristicas = extraer_caracteristicas_glcm(ventana, distances, angles)
        X.append(caracteristicas)
```

3.6.- Programe el clasificador para probar sus vectores de datos obtenidos. Utilizar al menos dos clasificadores, K-NN, Bayes o SVM (máquinas de soporte vectorial), de manera que pueda comparar sus resultados.

Al final usamos KNN y SVM y obtuvimos buenos resultados para las últimas configuraciones:
Se anexan los resultados en el siguiente inciso.

3.7.- Realice varias pruebas variando sus estadísticos o parámetros de descripción de características (features), así como el clasificador. Despliegue sus resultados.

Con las diferentes configuraciones obtuvimos estos resultados:
Además, se agregó un segmento de código para que cada vez que se clasifique se muestren 5 ventanas aleatorias con la etiqueta original y la que el modelo predijo.

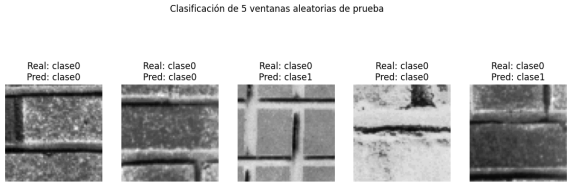
```
Resultados con KNN usando distancias: [1] y ángulos: [0]
              precision    recall  f1-score   support

      clase0       0.62      0.69      0.65        81
      clase1       0.60      0.56      0.58        62
      clase2       0.88      0.81      0.84        57

    accuracy                           0.69       200
   macro avg       0.70      0.69      0.69       200
weighted avg       0.69      0.69      0.69       200

Accuracy (KNN): 0.685
```

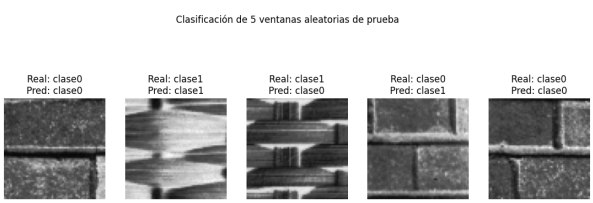
```
Resultados con SVM usando distancias: [1] y ángulos: [0]
              precision    recall  f1-score   support

      clase0       0.68      0.48      0.57        81
      clase1       0.69      0.98      0.81        62
      clase2       0.73      0.70      0.71        57

    accuracy                           0.70       200
   macro avg       0.70      0.72      0.70       200
weighted avg       0.70      0.70      0.68       200

Accuracy (SVM): 0.7
```

```
Resultados con KNN usando distancias: [1, 2, 3] y ángulos: [0, 0.7853981633974483, 1.5707963267948966]
              precision    recall  f1-score   support

      clase0       1.00      0.98      0.99        81
      clase1       0.98      1.00      0.99        62
      clase2       0.98      1.00      0.99        57

    accuracy                           0.99       200
   macro avg       0.99      0.99      0.99       200
weighted avg       0.99      0.99      0.99       200

Accuracy (KNN): 0.99
```

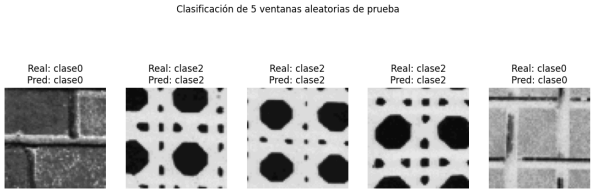
```
Resultados con SVM usando distancias: [1, 2, 3] y ángulos: [0, 0.7853981633974483, 1.5707963267948966]
              precision    recall  f1-score   support

      clase0       1.00      0.99      0.99        81
      clase1       1.00      1.00      1.00        62
      clase2       0.98      1.00      0.99        57

    accuracy                           0.99       200
   macro avg       0.99      1.00      1.00       200
weighted avg       1.00      0.99      1.00       200

Accuracy (SVM): 0.995
```

Las 2 imágenes inferiores son muy ilustrativas ya que hay instancias con clasificaciones erróneas, y era de esperarse con el accuracy de .68, está confundiendo los ladrillos con la palma.



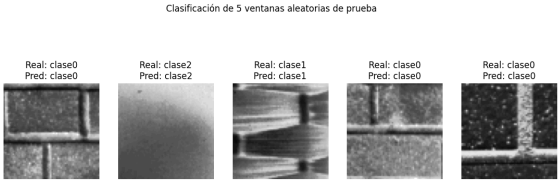


En esta nueva imagen que tiene los resultados del clasificador de 99% de accuracy vemos a todas las ventanas clasificadas correctamente. Recordando que son muestras al azar.



3.8.- Explicar por qué obtuvo los resultados y qué pasaría si se varía algún parámetro utilizado en el método.

Se obtuvieron estos resultados ya que los vectores de características de texturas semejantes igualmente son cercanos; y si le sumamos que generamos 100 instancias por imagen aproximadamente, tenemos bastantes datos para que el modelo pueda generalizar bien; esto por la parte de los estadísticos, por parte de la orientación y la distancia, aumentando significativamente el consumo computacional agregamos varias orientaciones a la vez y esto permitió que pudiera distinguir bien entre todas las clases.



IV. CONCLUSIONES

En conclusión, esta práctica fue muy ilustrativa para comenzar a tomar en cuenta cosas más allá del color para clasificar imágenes, como lo son las variaciones en tonos de gris que puede representarnos profundidad, bordes, etc, además, en esta ocasión ya lo hacemos también por regiones y facilita mucho el análisis y el procesamiento ya que con GLCM comparamos los pixeles con sus vecinos y esto nos puede ayudar a detectar algún tipo de patrón en estructuras más complejas.
Ahora, en comparación con el análisis que hicimos semanas atrás utilizando Bayes, donde nos enfocábamos principalmente en el color de los píxeles, tomando la media y la covarianza para clasificar cada píxel individualmente, el enfoque actual va un paso más allá. Anteriormente, la clasificación basada en color nos proporcionaba una buena base para problemas donde la diferencia en color era el principal diferenciador. Sin embargo, al incluir las relaciones espaciales entre píxeles a través de GLCM, podemos capturar patrones más complejos como la textura y las variaciones sutiles en la escala de grises, que son características cruciales en muchas imágenes.

La importancia de este enfoque en el área de clasificación de imágenes es fundamental, ya que mejora nuestra capacidad para distinguir entre diferentes clases que comparten características visuales similares, pero difieren en aspectos más complejos como la textura. En áreas como la visión por computadora, la segmentación de imágenes médicas, o el reconocimiento de objetos en imágenes, el análisis de texturas y patrones de co-ocurrencia abre nuevas posibilidades para aumentar la precisión y robustez de los modelos.

El paso de clasificar píxeles basándonos únicamente en el color a utilizar texturas y relaciones espaciales entre píxeles es un avance clave que fortalece los métodos de clasificación de imágenes. Nos permite abordar problemas más desafiantes donde el color no es el único atributo relevante, lo que resulta crucial en aplicaciones donde la información visual debe ser procesada en su totalidad para obtener resultados más precisos y útiles.
Finalmente, aunque en esta práctica no hemos variado los parámetros estadísticos, es importante destacar que las implicaciones de ajustar distintos estadísticos (como contraste, correlación o energía) y diferentes distancias entre píxeles pueden influir significativamente en los resultados de clasificación. Estas variaciones permiten afinar el análisis y la representación de texturas de una manera más precisa, lo que ofrece una gran flexibilidad al adaptar el modelo a distintos tipos de imágenes y objetivos. En conjunto, la combinación de estadísticos, distancias, y patrones de textura refuerza las capacidades de los sistemas de clasificación de imágenes.

V. CÓDIGOS

En este apéndice, se anexará el contenido de los códigos para los ejercicios desarrollados en la presente práctica.

```
import matplotlib.pyplot as plt
import random
import os
from skimage import io
from skimage.feature import graycomatrix,
graycoprops
import numpy as np
from sklearn.neighbors import
KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import
train_test_split
```

```
from sklearn.metrics import classification_report, accuracy_score

images_directory = 'Brodatz - copia'
imagenes = []
etiquetas = []

# Cargamos 4 imagenes de clase0 (ladrillo), 3 de clase 1 que son las palmas y 3 clase 2 (puntos)
clases = ['clase0', 'clase1', 'clase2']
for clase in clases:
    subdirectory = os.path.join(images_directory, clase)

    for cont, file_name in enumerate(sorted(os.listdir(subdirectory))):
        if cont >= 12:
            break

        file_path = os.path.join(subdirectory, file_name)

        if os.path.isfile(file_path):
            imagen = io.imread(file_path)
            imagenes.append(imagen)
            etiquetas.append(clase) #Aqui se hace el etiquetado dependiendo de la carpeta origen

# Ventana cuadrada de 64x64, las imagenes son de 640x640
ventanita = 64
ventanas = []
ventanas_etiquetas = []

# Recorrer las imágenes y dividirlas en ventanas
for cont, imagen in enumerate(imagenes):
    alto, ancho = imagen.shape[:2]
    for i in range(0, alto, ventanita):
        for j in range(0, ancho, ventanita):
            # Extraer ventana de la imagen
            ventana = imagen[i:i + ventanita, j:j + ventanita]
            # Asegurarse de que la ventana sea del tamaño correcto
            if ventana.shape[0] == ventanita and ventana.shape[1] == ventanita:
                ventanas.append(ventana)

                ventanas_etiquetas.append(etiquetas[cont])
                # Asignar la etiqueta


# Funcion del GLCM
def extraer_caracteristicas_glcm(ventana, distances, angles):
    # Convertir ventana a escala de grises si es una imagen RGB
    if len(ventana.shape) == 3:
        ventana_gris = np.dot(ventana[..., :3], [0.2989, 0.5870, 0.1140])
    else:
        ventana_gris = ventana

    caracteristicas = []

    for dist in distances:
        for angle in angles:
            glcm = graycomatrix(ventana_gris.astype(np.uint8), distances=[dist], angles=[angle],
                                levels=256, symmetric=True, normed=True)

            # Extraer características de GLCM
            contraste = graycoprops(glcm, 'contrast')[0, 0]
            homogeneidad = graycoprops(glcm, 'homogeneity')[0, 0]
            energia = graycoprops(glcm, 'energy')[0, 0]
            entropia = -np.sum(glcm * np.log2(glcm + (glcm == 0)))

            caracteristicas.extend([contraste, homogeneidad, energia, entropia])

    return caracteristicas


def evaluar_clasificadores(distances, angles):
    X = []

    for ventana in ventanas:
        caracteristicas = extraer_caracteristicas_glcm(ventana, distances, angles)
        X.append(caracteristicas)

    X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split(X, ventanas_etiquetas, test_size=0.2,

random_state=42)

    # KNN
    knn = KNeighborsClassifier(n_neighbors=3)
```

```
                    knn.fit(X_entrenamiento,
y_entrenamiento)
    y_pred_knn = knn.predict(X_prueba)

      print(f"\nResultados con KNN usando
distancias:    {distances}    y    ángulos:
{angles}")
      print(classification_report(y_prueba,
y_pred_knn))
                print(f"Accuracy      (KNN):
{accuracy_score(y_prueba, y_pred_knn)}")

    #  SVM
    svm = SVC(kernel='linear')
                    svm.fit(X_entrenamiento,
y_entrenamiento)
    y_pred_svm = svm.predict(X_prueba)

      print(f"\nResultados con SVM usando
distancias:    {distances}    y    ángulos:
{angles}")
      print(classification_report(y_prueba,
y_pred_svm))
                print(f"Accuracy      (SVM):
{accuracy_score(y_prueba, y_pred_svm)}")

    # Tomar 5 ventanas aleatoriamente para
ilustrar el programa
        fig,   axs   =   plt.subplots(1,   5,
figsize=(15, 5))
        fig.suptitle('Clasificación   de   5
ventanas aleatorias de prueba')

                  indices_aleatorios      =
random.sample(range(len(X_prueba)), 5)

                for      i,      idx      in
enumerate(indices_aleatorios):
                    ventana_original_idx   =
X.index(X_prueba[idx])  # Índice real de la
ventana en la lista original
                        ventana_original    =
ventanas[ventana_original_idx]    # Obtener
la ventana original
        etiqueta_real = y_prueba[idx]
        etiqueta_predicha = y_pred_knn[idx]

            axs[i].imshow(ventana_original,
cmap='gray')
                    axs[i].set_title(f'Real:
{etiqueta_real}\nPred:
{etiqueta_predicha}')
        axs[i].axis('off')

    plt.show()


configurations = [
    {'distancia': [1], 'orientacion': [0]},
    {'distancia': [3], 'orientacion': [0]},
    {'distancia': [1], 'orientacion': [np.pi
/ 4]},
     {'distancia': [1], 'orientacion': [3 *
np.pi / 4]},
    {'distancia': [1, 2], 'orientacion': [0,
np.pi / 4]},
      {'distancia': [1, 2], 'orientacion':
[np.pi / 2, 3 * np.pi / 4]},
    {'distancia': [1, 3], 'orientacion': [0,
np.pi / 4, np.pi / 2, 3 * np.pi / 4]},
    {'distancia': [1, 4], 'orientacion': [0,
np.pi / 2]},
     {'distancia': [1, 2, 3], 'orientacion':
[0, np.pi / 4, np.pi / 2]}
]

# Ciclo para mostrar las metricas
for config in configurations:
    distancias = config['distancia']
    angulos = config['orientacion']
          evaluar_clasificadores(distancias,
angulos)
```

References

[1] Sci-Kit Team. (2021). Scikit-Learn. About Us Recuperado el 7 de octubre de 2024, de https://scikit-learn.org/stable/about.html

[2] Matplotlib Dev Team (2022) Sobre Matplotlib, ¿Quienes Somos? Recuperado el 7 de octubre de 2024 https://matplotlib.org/stable/index.html

[3] Tuceryan, M., & Jain, A. K. (1993). Texture analysis. En Handbook of Pattern Recognition and Computer Vision (pp. 235–276). WORLD SCIENTIFIC.

[4] Satish, Anila & Sobana, K & B, Saranya. (2018). Fabric Texture Analysis and Weave Pattern Recognition by Intelligent Processing.