

# Práctica 3 parte dos: Caracterización de Texturas y GLCM de imágenes de patología

Serapio Hernández Alexis Arturo  
Facultad de Ingeniería  
UNAM  
Ciudad de México, México  
[alexis.serapio@ingenieria.unam.mx](mailto:alexis.serapio@ingenieria.unam.mx)

García López Erik  
Facultad de Ingeniería  
UNAM  
Ciudad de México, México  
[erikpumas999@gmail.com](mailto:erikpumas999@gmail.com)

Reyes Herrera Rogelio  
Facultad de Ingeniería  
UNAM  
Ciudad de México, México  
[masterfive5of@gmail.com](mailto:masterfive5of@gmail.com)

**Abstract**—In this article we explore texture characterization and classification techniques for detecting mastocytomas in tissue samples using digital image processing. Methods like GLCM and superpixel segmentation were applied, with classifiers such as KNN and K-Means used to identify healthy and affected regions. While GLCM captured detailed texture patterns, superpixels improved boundary precision, resulting in clearer classifications.

**Keywords**—*Basic Images Processing, Python, Pattern Recognition, KNN, Bayes, classifier, Texture analysis, GLCM, Superpixels, mastocitomas.*

## OBJETIVOS

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

## I. INTRODUCCIÓN

El análisis y la caracterización de texturas son esenciales en el procesamiento de imágenes, con aplicaciones diversas como la clasificación de superficies en imágenes satelitales y la detección de patrones en tejidos médicos. La textura se refiere a las variaciones espaciales de intensidad o color en una imagen, que revelan la estructura de los objetos. Para caracterizar estas texturas, se utilizan técnicas cuantitativas como las matrices de concurrencia, el análisis de frecuencia (con transformada de Fourier o análisis de Gabor) y los histogramas de gradientes.

Una vez caracterizadas, las texturas se clasifican mediante métodos como K-Nearest Neighbors (KNN), K-Means y máquinas de soporte vectorial (SVM).instancias, que asigna las clases de una muestra en función de las etiquetas de sus vecinos más cercanos.

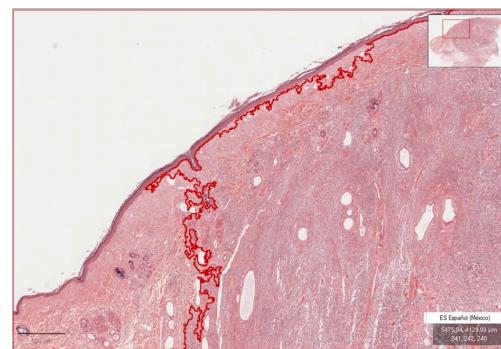
- **K-NN:** Es un enfoque basado en instancias que asigna la clase de una muestra en función de las etiquetas de sus vecinos más cercanos.

- **K-Means:** Es un algoritmo no supervisado que agrupa datos desconocidos, obteniendo clases por medio de la distancia de los datos al centroide.

Los mastocitos son un tipo de célula inmune que forma parte del sistema inmunológico y se encuentra principalmente en los tejidos conectivos, como la piel, los pulmones y el tracto digestivo. Estas células son clave en la defensa del cuerpo frente a patógenos y en la respuesta inflamatoria.

Los mastocitomas en cambio son crecimientos anormales causados por la proliferación excesiva de mastocitos en ciertos tejidos del cuerpo. Estos pueden ocurrir en la piel o en órganos internos y se desarrollan cuando los mastocitos se multiplican y se acumulan de manera descontrolada.

En esta práctica vamos a aplicar GLCM para hacer análisis en las imágenes digitales de las muestras obtenidas para poder clasificar las regiones en las que un animalito presente mastocitomas. El GLCM representa la frecuencia con la que ciertos pares de valores de intensidad ocurren en una imagen, separados por una distancia específica y en una dirección dada:



Para eliminar los bordes “cuadrados” o muy pixelados de nuestra clasificación vamos a apoyarnos en los superpixeles, estos son agrupaciones de pixeles en una imagen que tienen características similares que se tratan como unidades o

regiones homogéneas más grandes en lugar de píxeles individuales.

Estos agrupamientos permiten simplificar y reducir la cantidad de elementos a analizar, facilitando el procesamiento de la imagen.



Esta práctica entonces tiene como objetivos investigar y desarrollar técnicas de caracterización de texturas, aplicar los clasificadores mencionados y clasificar las imágenes muestras para determinar las regiones afectadas por los mastocitomas, ayudando por medio del reconocimiento de patrones al campo veterinario.

## II. DESARROLLO PRÁCTICA 1

### 3.1.- Análisis GLCM y recorrer toda una imagen que contenga al menos 2 regiones diferentes para poder clasificarlo.

Para realizarlo primero se instaló el software libre QuPath para abrir imágenes en formato .svs. Una vez dentro importamos la imagen “P19-2717B.svs” que era la más pesada de las 2 e importamos el archivo qpdata para tener la segmentación.

Tomamos capturas de pantalla de las regiones necesarias (zona sana, zona tumoral y ambas zonas) con ayuda de File > Export snapshot > Current viewer content (SVG).

Las imágenes nos las dio en formato SVG y en una página las convertimos a JPG para no tener que trabajar con el canal alpha.

Después de algunas pruebas con el código de la práctica pasada, elegimos KNN con 5 vecinos y la ventana de 256 para obtener los siguientes resultados preliminares:

93% de precisión

Resultados con KNN usando distancias: [1, 2, 3] y ángulos: [0, 0.7853981633974483, 1.5707963267948966]				
	precision	recall	f1-score	support
Tejido	0.98	0.88	0.93	59
Tumor	0.91	0.99	0.95	72
accuracy			0.94	131
macro avg	0.95	0.93	0.94	131
weighted avg	0.94	0.94	0.94	131
Accuracy (KNN):	0.9389312977099237			

Si usamos una ventana de 512 teníamos unas métricas del 100% pero eran muy pocas muestras, lo cual no es lo ideal.

Las ventanas de entrenamiento son de 256 y las de prueba de 128 para tener resultados más presentables,

A parte de las características que ya teníamos como contraste, homogeneidad, energía y entropía, también se agregaron las siguientes características:

```
dissimilarity      =      graycoprops(glcm,
'dissimilarity')[0, 0]
correlation        =      graycoprops(glcm,
'correlation')[0, 0]
asm = graycoprops(glcm, 'ASM')[0, 0]
```

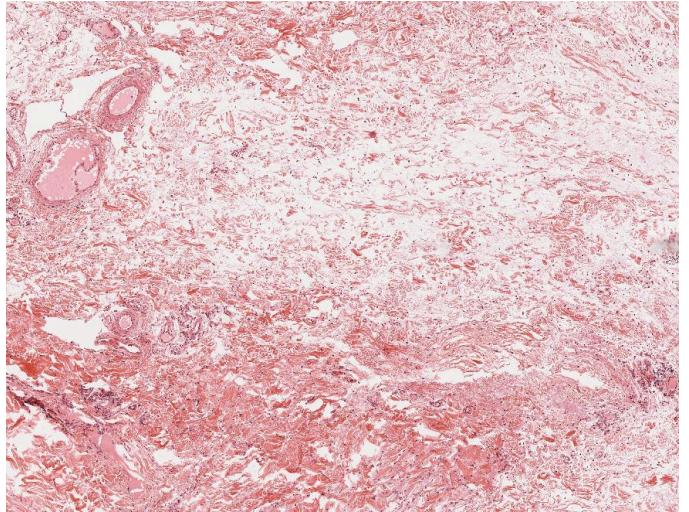
**Dissimilarity (Desemejanza):** Ayuda a captar diferencias de tono en regiones próximas.

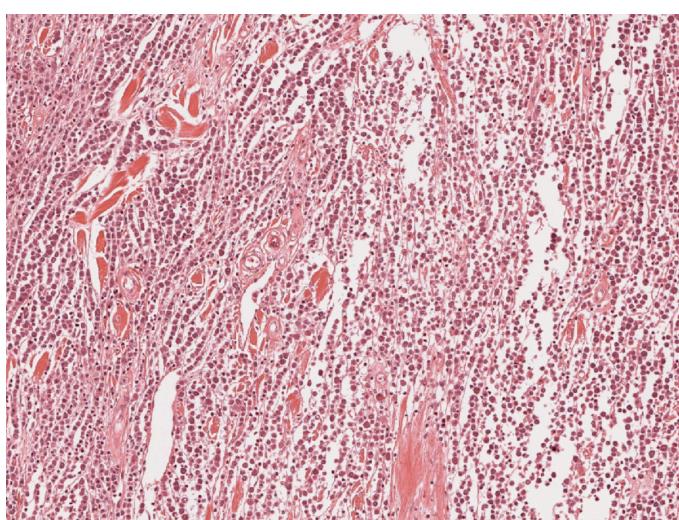
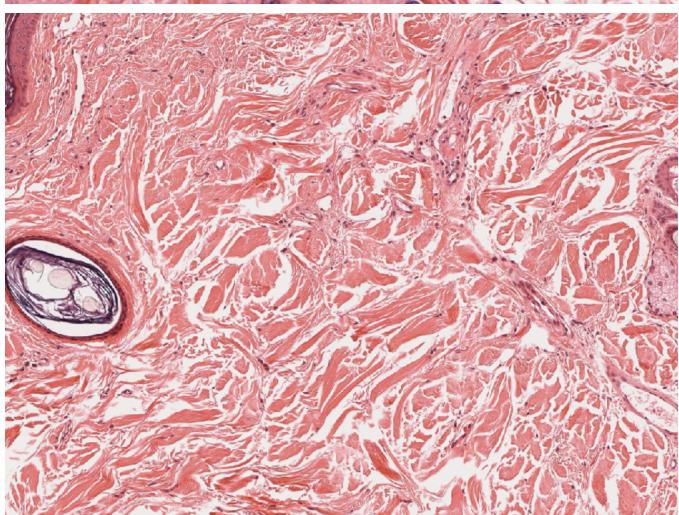
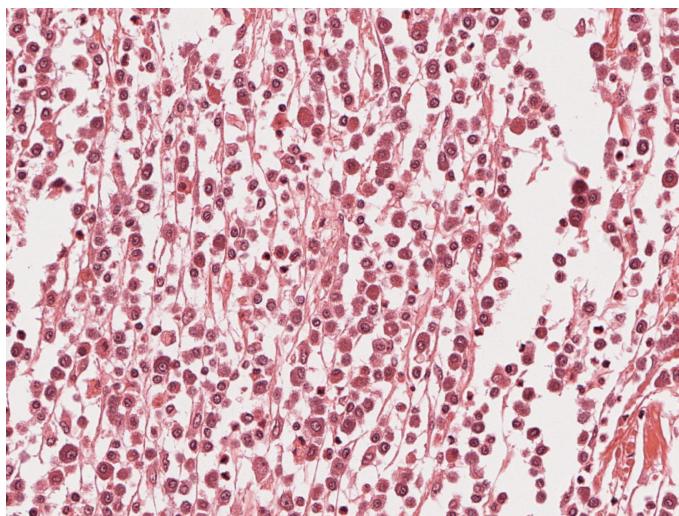
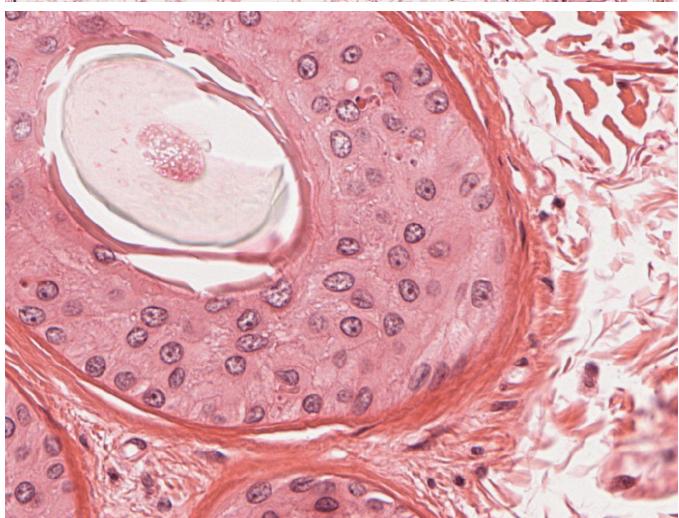
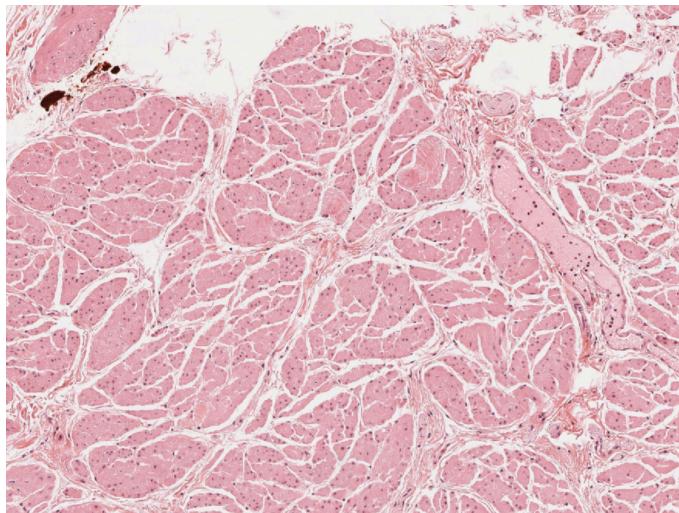
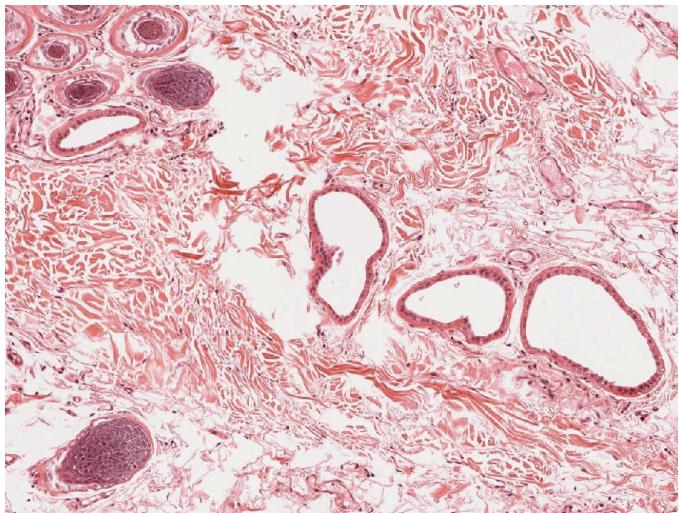
**ASM (Angular Second Moment) o Uniformidad:** similar a la energía, mide cuán homogénea es la textura, siendo útil para texturas suaves.

**Correlation (Correlación):** captura la relación entre los píxeles vecinos.

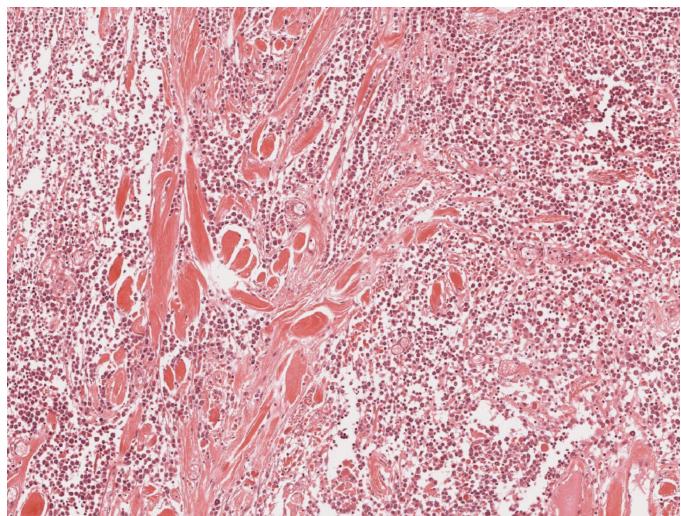
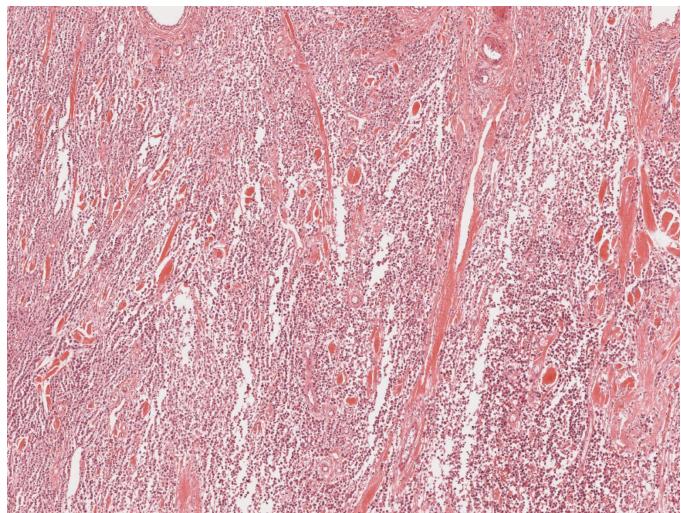
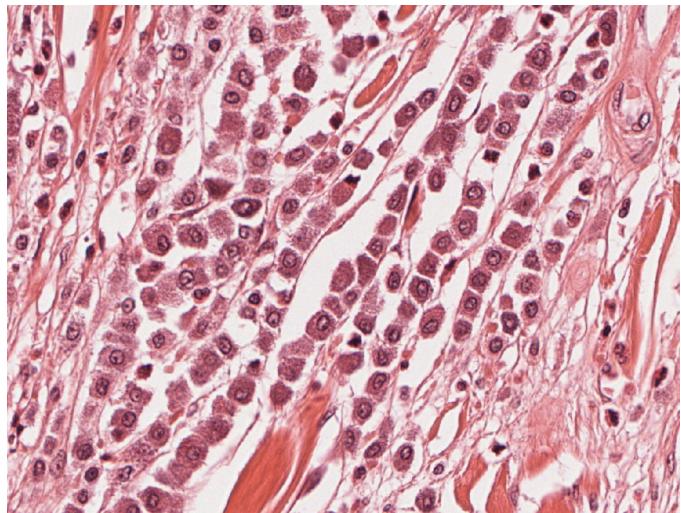
Ahora pasando al código modificado de esta práctica, tenemos las siguientes imágenes de entrenamiento a diferentes distancias para hacer más robusto el modelo:

**Sano (5 imágenes):**





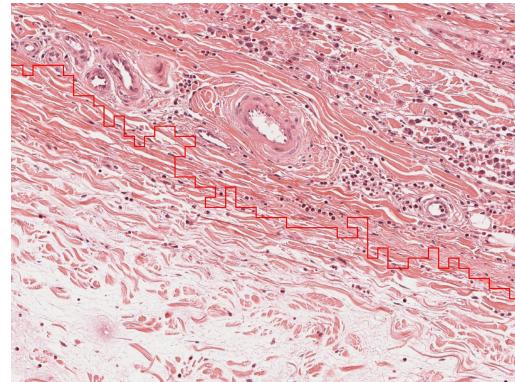
**Tumor (5 imágenes):**



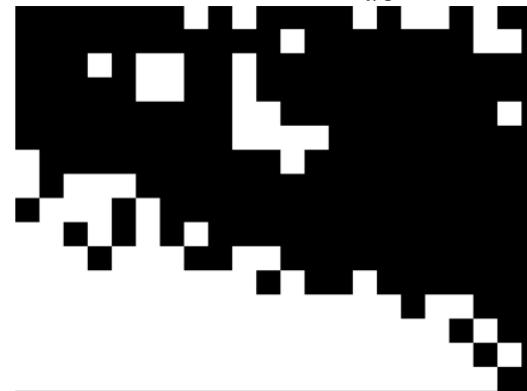
Todas las imágenes fueron divididas en ventanas y usadas en su totalidad para entrenamiento; por lo tanto al hacerle fit al modelo para entrenarlo, se usó ***test\_size=.01***. Dejando un

conjunto de pruebas despreciable y evaluando las métricas de entrenamiento en lugar de las de prueba.

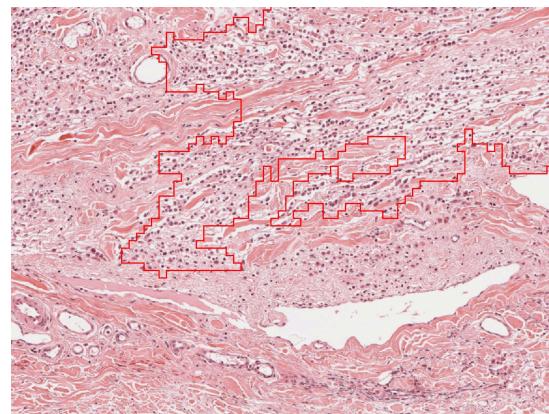
Pasando a los resultados tenemos lo siguiente, teniendo en cuenta que usamos 10 imágenes de entrenamiento y 3 de prueba:



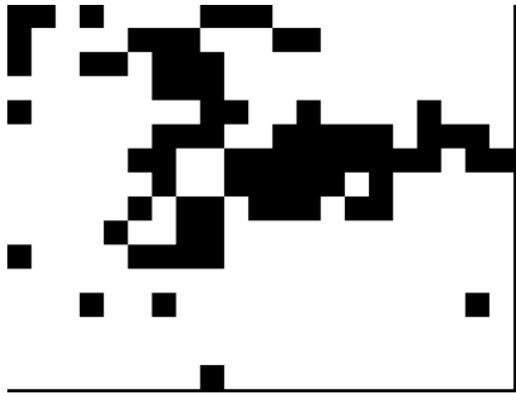
Clasificación de ambos1.jpg



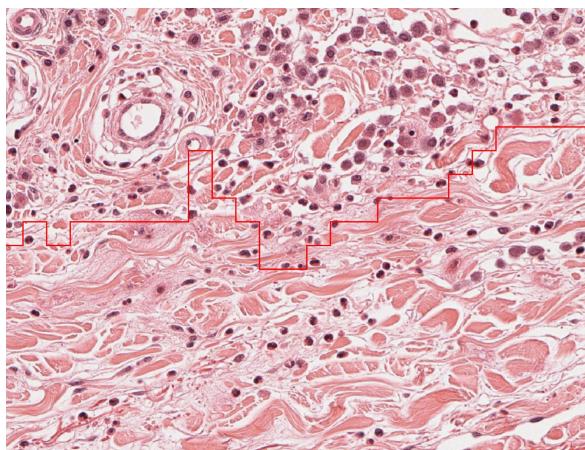
Los resultados son medianamente buenos para la primera imagen.



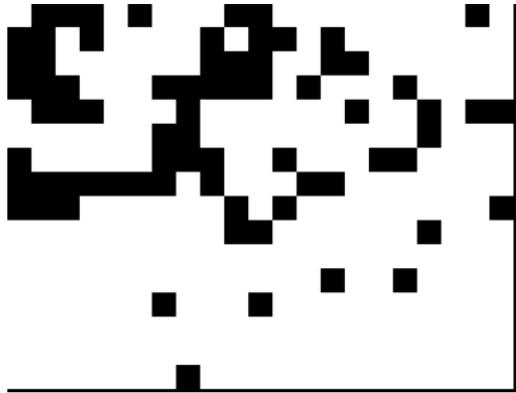
Clasificación de ambos2.jpg



En esta segunda muestra los aciertos bajan pero más o menos se visualiza la división entre la zona sana y la enferma.



Clasificación de ambos3.jpg



Mismo caso con esta imagen, se alcanza a distinguir que zona corresponde a que clase *pero está muy mal clasificada*.

### 3.2.- Análisis con superpixeles para comparar sus resultados con el anterior.

El código anterior no cambió demasiado, únicamente la función de extracción de características.

```
for cont, imagen in enumerate(imagenes):
    # Aplicar SLIC para segmentar la imagen
    etiquetas_superpixeles = slic(imagen, n_segments=n_segments, compactness=10, start_label=1)
    # Extraer características de los superpixeles
    características = extraer_características_superpixeles(imagen, etiquetas_superpixeles)
    ventanas.extend(características)
    ventanas_etiquetas.extend([etiquetas[cont]] * len(características))
```

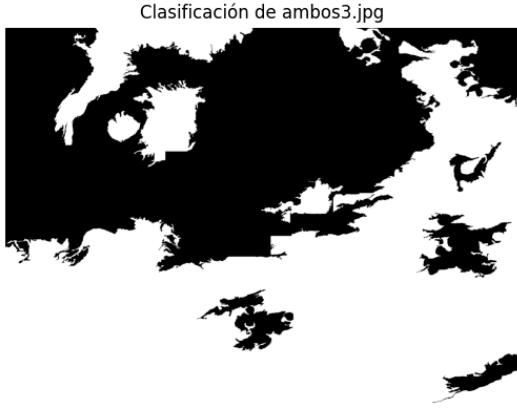
Estos cambios implican que nuestros vectores van a ser diferentes a los que generaba GLCM por lo tanto nuestro modelo “ideal” tuvo cambios “negativos en las métricas”, pero como demoraba mucho cada prueba decidimos proceder con el mismo, al final se trata de comparar los resultados.

Evaluación del modelo KNN en entrenamiento:				
	precision	recall	f1-score	support
Tejido	0.88	0.91	0.90	640
Tumor	0.89	0.86	0.88	543
accuracy			0.89	1183
macro avg	0.89	0.89	0.89	1183
weighted avg	0.89	0.89	0.89	1183

No bajó demasiado, así que los resultados son buenos.



Tenemos una imagen más suave y con mayor acierto en cuanto a la clasificación, incluso con la imagen que con GLCM salía pésimo:



Aunque aún no son los resultados esperados, mejoraron y se ven más cercanos a lo que esperábamos visualizar.

### III. CONCLUSIONES

En conclusión, esta práctica toca temas muy interesantes e importantes respecto a la tarea de clasificación en imágenes, específicamente en imágenes médicas; nosotros tomamos esta asignación como una parte introductoria al área y a nuevas herramientas para ir mejorando la segmentación de regiones diferentes. En este caso, inferimos que podemos mejorar el modelo si dedicamos algunos días variando los parámetros del algoritmo, y evidentemente si agregamos mas datos; es decir, mas imágenes con mas distancias; en este caso no lo hicimos ya que se demora mucho en procesar incluso con la poca información que tenemos, pero son cosas que tendrían un impacto positivo en el proceso.

Por último comparando los resultados, vimos una enorme mejora en los resultados con superpixeles respecto a GLCM, ya que esto suaviza la barrera y agrupa pixeles semejantes en zonas, lo cual hace que interpretar los resultados sea más sencillo y se pueda identificar mejor las zonas sin el sesgo del agrupamiento por ventanas cuadradas.

Sin embargo, el enfoque basado en GLCM sigue siendo valioso al capturar texturas y patrones de segundo orden, lo cual es esencial en el análisis de características sutiles en imágenes médicas. La combinación de ambos métodos amplía el potencial para lograr una segmentación más robusta y precisa en aplicaciones futuras.

### IV. CÓDIGOS

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

#### Código 1:

```
import matplotlib.pyplot as plt
import random
import os
from skimage import io
```

```
from skimage.feature import graycomatrix,
graycoprops, local_binary_pattern
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from sklearn.svm import SVC

images_directory = 'Patologia'
imagenes = []
etiquetas = []

# Cargar imágenes de entrenamiento y etiquetarlas según su carpeta de origen
clases = ['Tejido', 'Tumor']
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)

# Parámetros para las ventanas y las configuraciones de GLCM y LBP
ventanita_entrenamiento = 256
ventanita_prueba = 128
distancias = [1, 2, 3, 4, 5, 6, 7]
angulos = [0, np.pi / 4, np.pi / 6, np.pi / 3, np.pi / 2, 2 * np.pi / 3, 5 * np.pi / 6, 3 * np.pi / 4]
ventanas = []
ventanas_etiquetas = []

# Dividir imágenes de entrenamiento en ventanas de 256x256 y extraer características
def extraer_caracteristicas_glcm(ventana, distancias, angulos):
    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 distancias:
        for angle in angulos:
            caracteristicas.append(local_binary_pattern(ventana_gris, n=2, r=1, P=2, M=4, L=2, R=1, method='nri_uniform'))
```

```

glcm =
graycomatrix(ventana_gris.astype(np.uint8),
distances=[dist], angles=[angle],
levels=256,
symmetric=True, normed=True)
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)))
dissimilarity =
graycoprops(glcm, 'dissimilarity')[0, 0]
correlation = graycoprops(glcm,
'correlation')[0, 0]
asm = graycoprops(glcm,
'ASM')[0, 0]

características.extend([contraste,
homogeneidad, energía, entropia,
dissimilarity, correlation, asm])

ventana_gris_uint8 = (ventana_gris *
255).astype(np.uint8) if ventana_gris.dtype
!= np.uint8 else ventana_gris
lbp =
local_binary_pattern(ventana_gris_uint8,
P=8, R=1, method='uniform')
lbp_hist, _ = np.histogram(lbp.ravel(),
bins=np.arange(0, lbp.max() + 1),
density=True)
características.extend(lbp_hist)

return características

# Extraer ventanas y características para
entrenamiento
for cont, imagen in enumerate(imágenes):
    alto, ancho = imagen.shape[:2]
    for i in range(0, alto,
ventanita_entrenamiento):
        for j in range(0, ancho,
ventanita_entrenamiento):
            ventana = imagen[i:i +
ventanita_entrenamiento, j:j +
ventanita_entrenamiento]
            if ventana.shape[0] ==
ventanita_entrenamiento and
ventana.shape[1] ==
ventanita_entrenamiento:
                ventanas.append(extraer_características_glc
m(ventana, distancias, ángulos))
                ventanas_etiquetas.append(etiquetas[cont])

```

```

# Entrenar el modelo KNN
X_entrenamiento, X_prueba, y_entrenamiento,
y_prueba = train_test_split(
    ventanas, ventanas_etiquetas,
test_size=0.01, random_state=42
)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_entrenamiento, y_entrenamiento)
print("\nEvaluación del modelo KNN en
entrenamiento:")
print(classification_report(y_entrenamiento,
knn.predict(X_entrenamiento)))

# Cargar y procesar imágenes de prueba,
dividir en ventanas de 128x128
prueba_directory =
os.path.join(images_directory, 'Prueba')
for file_name in
sorted(os.listdir(prueba_directory)):
    file_path =
os.path.join(prueba_directory, file_name)
    if os.path.isfile(file_path):
        imagen_prueba = io.imread(file_path)
        alto, ancho =
imagen_prueba.shape[:2]
        resultado = np.zeros((alto, ancho),
dtype=np.uint8)

        # Clasificar cada ventana de prueba
        for i in range(0, alto,
ventanita_prueba):
            for j in range(0, ancho,
ventanita_prueba):
                ventana = imagen_prueba[i:i +
ventanita_prueba, j:j + ventanita_prueba]
                if ventana.shape[0] ==
ventanita_prueba and ventana.shape[1] ==
ventanita_prueba:
                    características =
extraer_características_glc(ventana,
distancias, ángulos)
                    predicción =
knn.predict([características])[0]

                    # Asignar colores según
la predicción
                    color = 255 if
predicción == 'Tejido' else 0
                    resultado[i:i +
ventanita_prueba, j:j + ventanita_prueba] =
color

                    # Mostrar la imagen clasificada en
blanco y negro
                    plt.imshow(resultado, cmap='gray')
                    plt.title(f'Clasificación de
{file_name}')
                    plt.axis('off')

```

```
plt.show()
```

**Código 2:**

```
import matplotlib.pyplot as plt
import random
import os
from skimage import io
from skimage.segmentation import slic
from skimage.color import label2rgb
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

images_directory = 'Patología'
imagenes = []
etiquetas = []

# Cargar imágenes de entrenamiento y etiquetarlas según su carpeta de origen
clases = ['Tejido', 'Tumor']
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)

# Parámetros para la segmentación SLIC
n_segments = 200 # Número de superpíxeles a crear
ventanas = []
ventanas_etiquetas = []

# Extraer características de los superpíxeles
def extraer_caracteristicas_superpixeles(imagen, etiquetas):
    caracteristicas = []
    for label in np.unique(etiquetas):
        mask = (etiquetas == label)
        if np.sum(mask) > 0:
            region = imagen[mask]
            mean_color = np.mean(region, axis=0)
```

```
caracteristicas.append(mean_color)
    return caracteristicas

# Procesar cada imagen
for cont, imagen in enumerate(imagenes):
    # Aplicar SLIC para segmentar la imagen
    etiquetas_superpixeles = slic(imagen, n_segments=n_segments, compactness=10, start_label=1)
        # Extraer características de los superpíxeles
        caracteristicas = extraer_caracteristicas_superpixeles(imagen, etiquetas_superpixeles)
        ventanas.extend(caracteristicas)

    ventanas_etiquetas.extend([etiquetas[cont]] * len(caracteristicas))

# Entrenar el modelo KNN
X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split(
    ventanas, ventanas_etiquetas, test_size=0.01, random_state=42)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_entrenamiento, y_entrenamiento)
print("\nEvaluación del modelo KNN en entrenamiento:")
print(classification_report(y_entrenamiento, knn.predict(X_entrenamiento)))

# Cargar y procesar imágenes de prueba
prueba_directory = os.path.join(images_directory, 'Prueba')
for file_name in sorted(os.listdir(prueba_directory)):
    file_path = os.path.join(prueba_directory, file_name)
    if os.path.isfile(file_path):
        imagen_prueba = io.imread(file_path)

        # Aplicar SLIC a la imagen de prueba
        etiquetas_superpixeles = slic(imagen_prueba, n_segments=n_segments, compactness=10, start_label=1)

        resultado = np.zeros(imagen_prueba.shape[:2], dtype=np.uint8)

        # Clasificar cada superpíxel
        for label in np.unique(etiquetas_superpixeles):
            mask = (etiquetas_superpixeles == label)
            if np.sum(mask) > 0:
```

```

        region = imagen_prueba[mask]
        mean_color = np.mean(region,
axis=0)
                caracteristicas =
mean_color.reshape(1, -1)
                prediccion =
knn.predict(caracteristicas)[0]

        # Asignar colores según la
predicción
        color = 255 if prediccion ==
'Tejido' else 0
        resultado[mask] = color

        # Mostrar la imagen clasificada en
blanco y negro
        plt.imshow(resultado, cmap='gray')
        plt.title(f'Clasificación de
{file_name}')
        plt.axis('off')
        plt.show()

```

## 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.
- [5] Concejo, J. A. (2021, abril 14). Mastocitoma canino: Diagnóstico y tratamiento para tu perro \*. Clínica veterinaria Madrid. <https://clinicaveterinariamadrid.es/2021/04/14/mastocitoma-canino-perro/>
- [6] Superpixels. (s/f). Mathworks.com. Recuperado el 28 de octubre de 2024, de [https://www.mathworks.com/help/images/ref/superpixels\\_es.html](https://www.mathworks.com/help/images/ref/superpixels_es.html)