UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

KS4BONES

Manual técnico

Sistema de detección de osteoporosis basada en el análisis estadístico de Kolmogórov-Smirnov

Nombre	Matricula	Carrera
Nicolas Mauricio Cantú Salinas	2013576	ITS
Aldair Alejandro Beltrán Meléndez	1802548	ITS
Rodrigo Lerma de la Garza	2132060	ITS
Juan de Dios Moreno Rivero	2132187	ITS

Dirección
Ciudad Universitaria, 66455 San Nicolás de los Garza, N.L.

Datos
Grupo: 003
Equipo: 01

Contenido

INTRODUCCIÓN

Descripción del problema

La osteoporosis es una enfermedad esquelética caracterizada por la disminución progresiva de la densidad ósea y el deterioro de la microestructura del tejido trabecular, lo que incrementa significativamente el riesgo de fracturas, especialmente en el radio distal. Aunque la prueba estándar para su diagnóstico es la densitometría ósea (DXA), su disponibilidad y costo limitan su acceso en muchos contextos.

Esta enfermedad se divide en normal, osteopenia y osteoporosis se refieren a la densidad mineral ósea, La densidad mineral ósea indica la fortaleza de los huesos. La normalidad se considera un nivel de densidad ósea adecuado, la osteopenia una densidad más baja de lo normal, y la osteoporosis una densidad muy baja, lo que aumenta el riesgo de fracturas.

Una alternativa viable y accesible es el análisis de imágenes radiográficas convencionales. No obstante, la evaluación visual de las estructuras óseas en estas imágenes es altamente subjetiva y dependiente del observador. En respuesta a este problema, este proyecto propone un enfoque automatizado y estadísticamente fundamentado para la detección de posibles indicios de los diferentes niveles de osteoporosis, mediante la comparación de distribuciones de intensidad en regiones trabeculares específicas utilizando el test de Kolmogórov-Smirnov (KS).

Dado el alcance académico del proyecto y la imposibilidad de contar con un conjunto de datos clínicos reales, se opta por trabajar con radiografías simuladas, generadas a partir de modelos computacionales que representan distintos grados de deterioro óseo. Esto permite validar la viabilidad técnica del enfoque en un entorno controlado.

Objetivos del algoritmo

El objetivo principal del algoritmo es detectar alteraciones en los patrones trabeculares de radiografías simuladas de muñeca, utilizando el test de Kolmogórov-Smirnov como herramienta estadística para comparar distribuciones de intensidad dentro de regiones óseas específicas.

Los objetivos específicos son:

- Generar sets de imágenes basadas en parámetros que definan grados de deterioro óseo.
- Extraer histogramas de intensidad de regiones trabeculares y corticales simuladas.
- Aplicar la prueba KS para comparar dichas distribuciones en el set generado.
- Calcular métricas como el estadístico **D** y su **p-value**.

Alcance y limitaciones

Este proyecto tiene como **alcance** principal el desarrollo y prueba de un prototipo conceptual para la detección de osteoporosis utilizando datos sintéticos. Se centra exclusivamente en:

- Radiografías simuladas en base a regiones trabeculares y corticales
- Comparaciones estadísticas mediante KS
- Evaluación de la metodología en un entorno controlado

Las **limitaciones** incluyen:

- No se utilizan datos clínicos reales, por lo tanto, los resultados no tienen validez diagnóstica.
- El modelo no contempla variabilidad anatómica extrema ni artefactos de imagen comunes en estudios clínicos.
- El enfoque está diseñado para pruebas sintéticas y requiere adaptación previa a su uso en contextos clínicos reales.

Audiencia objetivo

Este trabajo está dirigido a estudiantes, docentes e investigadores del área de bioingeniería, procesamiento de imágenes médicas y análisis estadístico, interesados en métodos computacionales para diagnóstico asistido. También puede ser de utilidad para profesionales de ciencias de la salud interesados en herramientas de apoyo basadas en inteligencia artificial o pruebas estadísticas aplicadas a imágenes médicas.

REQUISITOS Y ESPECIFICACIONES

Requisitos del sistema

Versión de Python requerida

- **Python 3.13.3 o superior**
  Se recomienda Python 3.10 para mejor compatibilidad con paquetes científicos y de análisis de datos.

Librerías y dependencias

Las siguientes librerías son necesarias para el funcionamiento del sistema. Están incluidas en el archivo requirements.txt y pueden instalarse automáticamente.

Librería	Versión requerida	Propósito
numpy	>=1.20.0	Cálculo numérico y manejo de arrays
opencv-python	>=4.5.0	Procesamiento y manipulación de imágenes
Pillow	>=8.0.0	Carga y edición de imágenes
matplotlib	>=3.4.0	Visualización de imágenes y resultados
tk	>=0.1.0	Interfaz gráfica con Tkinter
psutil	==7.0.0	Monitoreo de recursos del sistema
pyinstaller	==6.13.0	Empaquetado del proyecto en un ejecutable (.exe)

Hardware mínimo recomendado

Recurso	Mínimo	Recomendado
CPU	2 núcleos	4 núcleos
RAM	4 GB	8 GB
Almacenamiento	500 MB libres	600+ MB libres
Resolución de pantalla	1280x720	1920x1080

Especificaciones técnicas

Complejidad computacional

- **Análisis KS por par de imágenes**:
  - Tiempo: O(n log n) para ordenar muestras
  - Evaluación Q_KS: O(k) donde k es el número de términos de la serie (por defecto k=100)
  -
- **Procesamiento total por lote:**
  - Si hay N pares de imágenes, la complejidad global es O(N·n log n)

Uso de memoria

- **Carga de imagen**: Aproximadamente ancho × alto × 1 byte (escala de grises)
- **Histograma y CDF**: Consumo adicional marginal (256 bins por imagen)
- En uso estándar (imágenes < 1024x1024), el consumo por análisis es menor a 100 MB.

Restricciones de entrada

- Se recomienda utilizar **imágenes en escala de grises** (.png, .jpg, .jpeg)
  - En el caso de no serlo, se aplicará un algoritmo de conversión.

- Tamaño recomendado: hasta **1024x1024** píxeles

- Imágenes deben tener el **mismo tamaño** para comparación válida
  - En el caso de no serlo, se aplicará un ajuste de dimensiones lo cual puede afectar a los resultados específicos del usuario.

- Las rutas de entrada deben ser válidas y accesibles

- No se admiten imágenes con canal alfa (transparencia)

DISEÑO DEL ALGORITMO

Descripción conceptual

El algoritmo propuesto tiene como objetivo estimar alteraciones del grado de deterioro óseo mediante la comparación estadística de distribuciones de grises extraídas de imágenes simuladas del radio distal. La base del procedimiento es la prueba no paramétrica de Kolmogorov–Smirnov (KS), la cual permite comparar dos funciones de distribución acumulada (FDA) sin suponer una forma específica para las distribuciones.

Se parte de un conjunto de imágenes generadas sintéticamente bajo diferentes condiciones óseas: normal, osteopenia y osteoporosis. Centradas en el hueso trabecular y el tejido simulado. Luego se calcula la función de distribución acumulada de grises.

El algoritmo compara estadísticamente estas distribuciones utilizando la prueba KS. Los valores del estadístico y el valor-p se emplean para construir un índice compuesto que resume la similitud entre una imagen de prueba y las clases de referencia. Esta métrica se interpreta como una medida de distancia relativa al estado óseo de referencia más cercano, permitiendo una clasificación automática del caso analizado.

Fundamentos matemáticos

Distribución de grises acumulada

Dada una imagen en escala de grises con niveles de intensidad $g \in [0,255]$, el histograma normalizado h_g representa la probabilidad de que un píxel tome el valor g. A partir de éste, se construye la función de distribución acumulada (FDA) empírica como:

$$F_g = \sum_{g'=0}^{g} h_{g'}$$

Esta función es monótonamente no decreciente, y facilita comparaciones visuales y estadísticas entre imágenes, superando las limitaciones del histograma cuando se requiere superponer múltiples curvas.

Prueba de Kolmogórov-Smirnov

La prueba KS permite comparar dos FDA, $F_g^{(1)} - F_g^{(2)}$, correspondientes a dos imágenes diferentes. El estadístico KS se define como:

$$\widehat{D} = \max\left|F_g^{(1)} - F_g^{(2)}\right|$$

A partir de este valor se calcula el parámetro λ_{KS}:

$$\lambda_{KS} = \widehat{D} \cdot \sqrt{J + \frac{0.11}{\sqrt{J + 0.12}}}$$

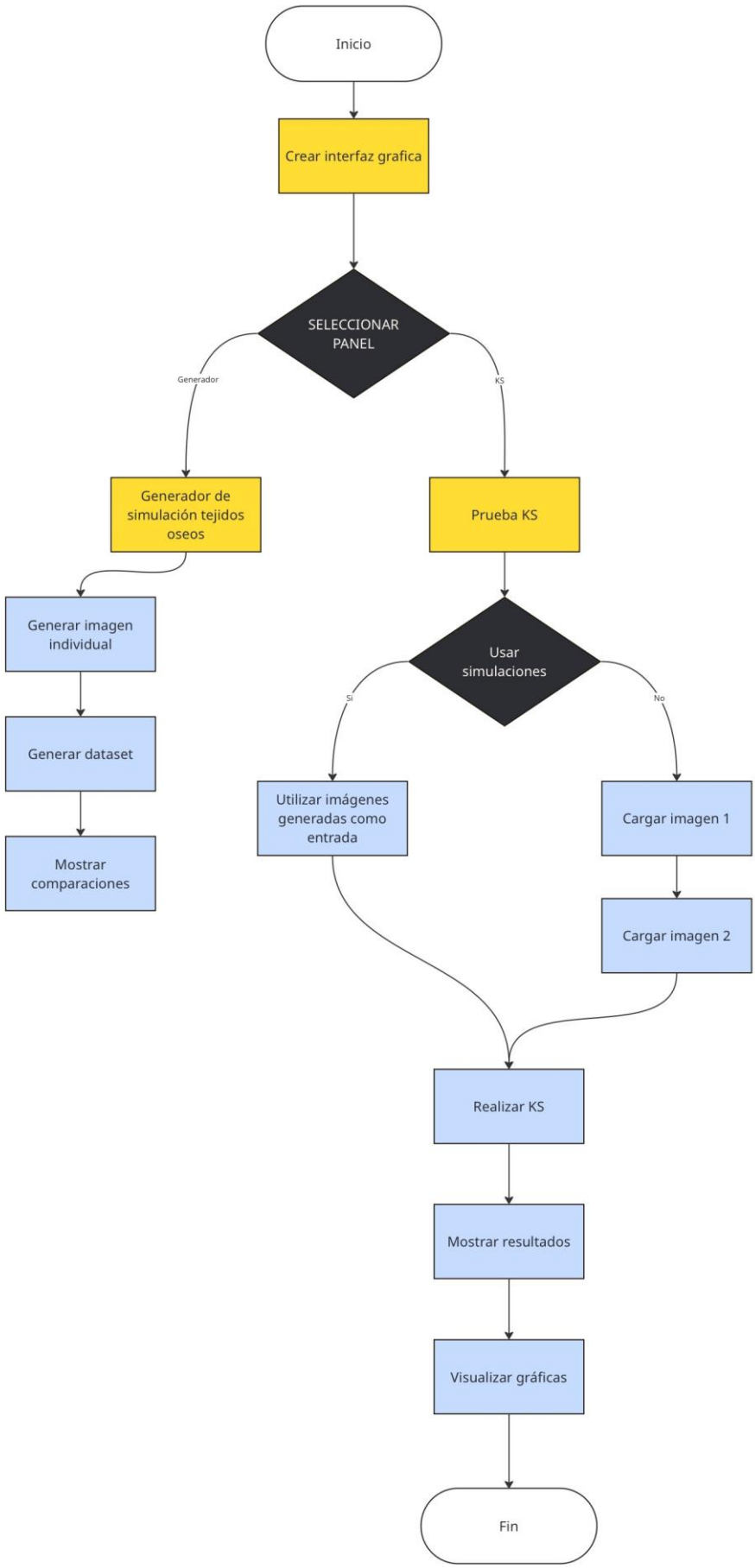
donde:

$$J = \frac{P_1 Q_1 \cdot P_2 Q_2}{P_1 Q_1 + P_2 Q_2}$$

El valor-p asociado a λ_{KS} se obtiene mediante la función:

$$Q_{KS}(\lambda) = 2 \sum_{j=1}^{\infty} (-1)^{j-1} \cdot e^{-2j^2\lambda^2}$$

Diagrama de flujo



Pseudocódigo

INICIO APLICACIÓN

INICIALIZAR ventana principal con tamaño 1000x700
CONFIGURAR fondo de la UI

INICIALIZAR variables:
  imagen1 ← None
  imagen2 ← None
  resultados_ks ← None

LLAMAR crear_interfaz():
  CREAR pestañas:
    - Simulación
    - Análisis KS

  EN pestaña Simulación:
    MOSTRAR opciones:
      - Generar imagen individual (normal o osteoporótica)
      - Generar dataset completo
    ASIGNAR funciones a botones

  EN pestaña Análisis KS:
    MOSTRAR opciones:
      - Cargar dos imágenes desde archivo
      - Generar automáticamente imágenes para análisis
      - Comparar tejidos normales vs osteoporóticos
      - Ejecutar análisis Kolmogorov-Smirnov
    ASIGNAR funciones a botones

FUNCIONES CLAVE:

```
generar_imagen_individual(tipo):
   imagen ← llamar a generadorTejidos.generar_tejido(tipo)
   mostrar_imagen_simulada(imagen, "Imagen generada")

generar_dataset(cantidad, tipo):
   dataset ← llamar a generadorTejidos.generar_dataset(cantidad, tipo)
   guardar dataset o procesar según necesidad

cargar_imagen(numero):
   abrir diálogo de archivo
   leer imagen
   asignar a imagen1 o imagen2 según 'numero'

generar_imagenes_para_ks():
   imagen1 ← generadorTejidos.generar_tejido("normal")
   imagen2 ← generadorTejidos.generar_tejido("osteoporotico")
   mostrar_imagenes_ks()

comparar_tipos():
   crear imágenes simuladas de ambos tipos
   mostrar comparación visual

realizar_analisis_ks():
   resultados_ks ← ks.realizar_prueba_ks(imagen1, imagen2)
   mostrar_resultados_ks(resultados_ks)

mostrar_resultados_ks(resultados):
   imprimir o visualizar:
      - estadístico KS
      - p-value
      - interpretación
```

FIN APLICACIÓN

Casos especiales

Lambda ≤ 0

Significado:

El valor de λ_KS (lambda_val) representa la distancia estadística entre las dos distribuciones de las imágenes. Si λ ≤ 0, significa que la distancia es nula o negativa (por definición no debería ser negativa, pero el código contempla ese caso para seguridad). Esto ocurre cuando las dos imágenes comparadas tienen distribuciones idénticas o prácticamente iguales.

Implicaciones en el análisis:

El valor-p devuelto es 1.0, indicando máxima similitud (no se rechaza la hipótesis nula).
Puede utilizarse para detectar si la comparación entre imágenes es redundante o si se está evaluando una imagen contra sí misma.
Este caso debe manejarse para evitar cálculos innecesarios o para indicar explícitamente que no hay diferencia entre las imágenes.

IMPLEMENTACIÓN

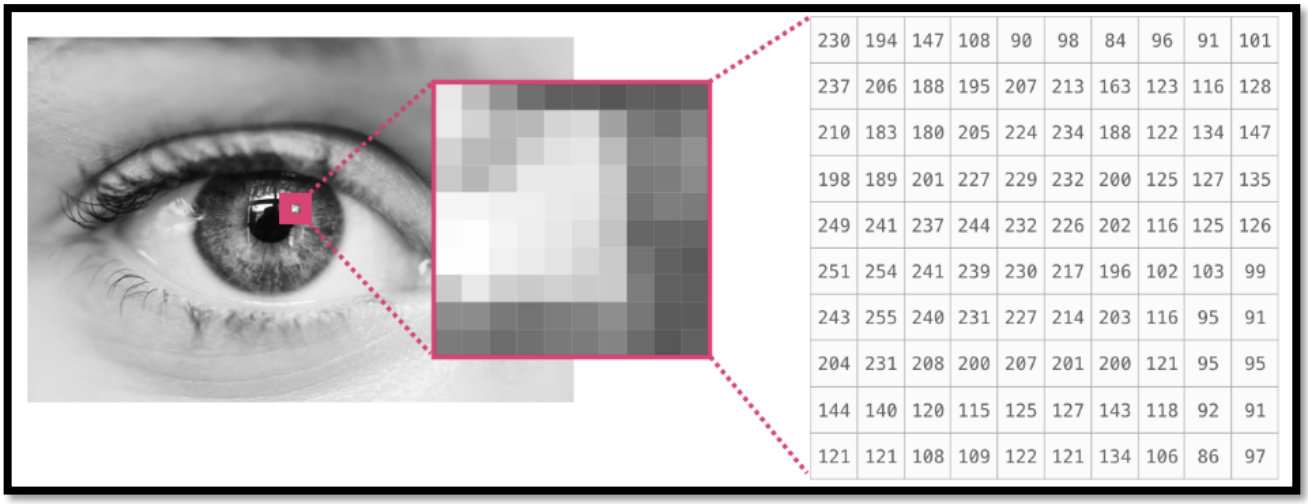
Arquitectura del código

El sistema está estructurado en módulos que permiten realizar el análisis estadístico de imágenes mediante la prueba de Kolmogórov-Smirnov y la generación/simulación de imágenes para comparación.

- **Módulo principal (DetectorOsteoporosisApp)**:
  Implementa la interfaz gráfica de usuario (UI), controla el flujo de ejecución y coordina la interacción entre componentes.

- **Módulo de análisis estadístico (ks.py)**:
  Contiene funciones para calcular el valor-p de la prueba KS, incluyendo el cálculo de λ_KS y la función Q_KS.

- **Módulo de generación de imágenes (generadorTejidos.py)**:
  Se encarga de crear imágenes simuladas de tejidos óseos con parámetros configurables para las pruebas.



Código fuente principal

Función principal

La función principal es el método **__init__** de la clase **DetectorOsteoporosisApp**, que inicializa la ventana principal, variables, y llama a los métodos para crear la interfaz.
El ciclo de eventos de la aplicación se gestiona mediante la librería Tkinter, que mantiene la aplicación activa y escucha eventos del usuario.

Funciones auxiliares

Dentro del proyecto existen varias funciones auxiliares que realizan tareas específicas, entre las más relevantes:

- calcular_lambda_ks(imagen1, imagen2): Calcula el valor λ_KS entre dos distribuciones de imágenes.
- calcular_q_ks(lambda_val): Calcula el valor-p a partir de λ_KS utilizando la serie infinita de la función Q_KS, con criterio de convergencia y manejo de casos especiales (e.g., lambda ≤ 0).
- generar_imagen_simulada(parametros): Genera una imagen de tejido óseo sintético según parámetros específicos.
- Funciones para cargar y mostrar imágenes en la interfaz.
- Funciones para actualizar elementos visuales según los resultados del análisis.

Clases y estructuras de datos

- **DetectorOsteoporosisApp**: Clase principal que representa la aplicación y contiene la lógica del UI y coordinación.

- **Estructuras de datos utilizadas**:
  - Arrays NumPy para representar imágenes y datos numéricos.
  - Listas y diccionarios para almacenar resultados y configuraciones.

No se utilizan estructuras complejas más allá de estas, dado el enfoque en procesamiento de imágenes y estadística.

Nota: El código fuente del proyecto esta en Anexos.

Comentarios y documentación interna

- Cada función contiene docstrings claros que describen propósito, argumentos y valores de retorno.

- Se documentan casos especiales, como el manejo de $\lambda \leq 0$ para evitar errores y alertar sobre imágenes idénticas.

- El código está comentado en las secciones críticas para facilitar mantenimiento y futuras extensiones.

Estilo de codificación

- El código sigue el estándar PEP 8 para Python:

    - Nombres de funciones en snake_case.
    - Uso de indentación de 4 espacios.
    - Líneas de código no superiores a 79 caracteres (en la medida de lo posible).

- Variables con nombres descriptivos para mejorar legibilidad.
- Separación clara entre bloques funcionales y visuales.
- Uso moderado y claro de excepciones y manejo de errores para robustez.

GUÍA DE INSTALACIÓN Y USO

Instalación

Descarga e instalación de dependencias

1. **Clonar o descargar el repositorio**:
   Descarga el código fuente desde el repositorio oficial o recibe la carpeta con los archivos.

2. **Crear un entorno virtual (recomendado)**:
   Para evitar conflictos con otras librerías de Python instaladas en el sistema, se recomienda crear un entorno virtual. En la terminal o consola, navega a la carpeta del proyecto y ejecuta:

```
python3 -m venv venv
```

3. **Activar el entorno virtual**:

```
.\venv\Scripts\activate          # Windows
or
source venv/bin/activate         # macOS/Linux
```

   Esto crea una carpeta venv con un entorno Python aislado.

4. **Instalar dependencias**:
   El proyecto incluye un archivo requirements.txt con las librerías necesarias.
   Ejecuta:

```
pip install -r requirements.txt
```

Instrucciones de uso

Interfaz de línea de comandos

KS:

```
Valor-p calculado: 0.000000
=== RESULTADOS PRUEBA KOLMOGOROV-SMIRNOV ===
Estadístico D^: 0.207300
Lambda KS: 14.683522
Valor-p: 0.000000
Nivel de significancia: 0.05
Rechazar H0: True
Conclusión: Las imágenes tienen distribuciones diferentes
```

Generador de tejidos:

```
=== SIMULADOR DE IMÁGENES DE OSTEOPOROSIS ===

Imagen normal:
  - Dimensiones: (128, 128)
  - Rango intensidades: [104, 233]
  - Intensidad promedio: 171.8

Imagen osteopenia:
  - Dimensiones: (128, 128)
  - Rango intensidades: [74, 203]
  - Intensidad promedio: 139.3

Imagen osteoporosis:
  - Dimensiones: (128, 128)
  - Rango intensidades: [44, 173]
  - Intensidad promedio: 109.2

Generando dataset de prueba...
Generando 5 imágenes de tipo normal...
Generando 5 imágenes de tipo osteopenia...
Generando 5 imágenes de tipo osteoporosis...
Dataset normal: (5, 64, 64)
Dataset osteopenia: (5, 64, 64)
Dataset osteoporosis: (5, 64, 64)
```

VALIDACIÓN Y PRUEBAS

Metodología de pruebas

El sistema implementa un enfoque de pruebas multicapa que garantiza la confiabilidad del análisis:

1. **Pruebas de Unidad:** Validación individual de cada función
2. **Pruebas de Integración:** Verificación del flujo completo
3. **Pruebas de Validación:** Comparación con estándares conocidos
4. **Pruebas de Rendimiento:** Evaluación de eficiencia computacional

Casos de prueba

Pruebas unitarias

Objetivo

Las pruebas unitarias verifican el correcto funcionamiento de los componentes individuales del sistema de detección de osteoporosis.

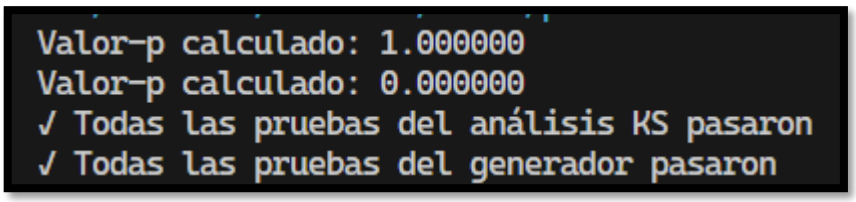
Test 1: Validación del Generador de Tejidos

```
def test_generador_tejidos():
    """Prueba unitaria para el generador de tejidos"""

    # Test 1: Verificar dimensiones
    imagen = generar_imagen_hueso_simulada(width=100, height=150)
    assert imagen.shape == (150, 100), "Dimensiones incorrectas"

    # Test 2: Verificar rango de valores
    assert imagen.min() >= 0 and imagen.max() <= 255, "Valores fuera de rango [0,255]"

    # Test 3: Verificar diferencias entre tipos
    img_normal = generar_imagen_hueso_simulada(tipo=TipoTejido.NORMAL, seed=1)
    img_osteo = generar_imagen_hueso_simulada(tipo=TipoTejido.OSTEOPOROSIS,
seed=1)

    # Debería haber diferencia significativa en intensidad promedio
    diff_promedio = abs(img_normal.mean() - img_osteo.mean())
    assert diff_promedio > 20, "Diferencia insuficiente entre tipos de tejido"

    print("✓ Todas las pruebas del generador pasaron")

    test_generador_tejidos()
```

Test 2: Validación del Análisis KS

```
def test_analisis_ks():
    """Prueba unitaria para el análisis Kolmogorov-Smirnov"""

    # Test con imágenes idénticas (no debería rechazar H0)
    imagen_base = np.random.randint(0, 256, (50, 50), dtype=np.uint8)
    resultado_identica = prueba_ks_imagenes(imagen_base, imagen_base.copy())

    assert not resultado_identica['rechazar_H0'], "Error: rechaza H0 con imágenes idénticas"
    assert resultado_identica['estadistico_D'] < 0.01, "Estadístico D muy alto para imágenes idénticas"

    # Test con imágenes muy diferentes
    img1 = np.full((50, 50), 50, dtype=np.uint8)  # Imagen uniforme oscura
    img2 = np.full((50, 50), 200, dtype=np.uint8)  # Imagen uniforme clara

    resultado_diferente = prueba_ks_imagenes(img1, img2)
    assert resultado_diferente['rechazar_H0'], "Error: no rechaza H0 con imágenes muy diferentes"

    print("✓ Todas las pruebas del análisis KS pasaron")

test_analisis_ks()
```

Output:



Pruebas de integración

Descripción

El test de integración verifica la funcionalidad completa del sistema, evaluando la generación de imágenes sintéticas y el análisis estadístico mediante el test de Kolmogórov-Smirnov (KS).

Componentes Evaluados

1. Generación de Imágenes
   - Genera imágenes sintéticas para tres tipos de tejido óseo
   - Dimensiones de prueba: 64x64 píxeles
   - Usa semilla fija (42) para reproducibilidad

2. Análisis KS
   - Test 1: Compara tejido normal vs osteoporosis
   - Test 2.1: Compara tejido normal consigo mismo
   - Test 2.2: Valida consistencia comparando normal vs osteoporosis

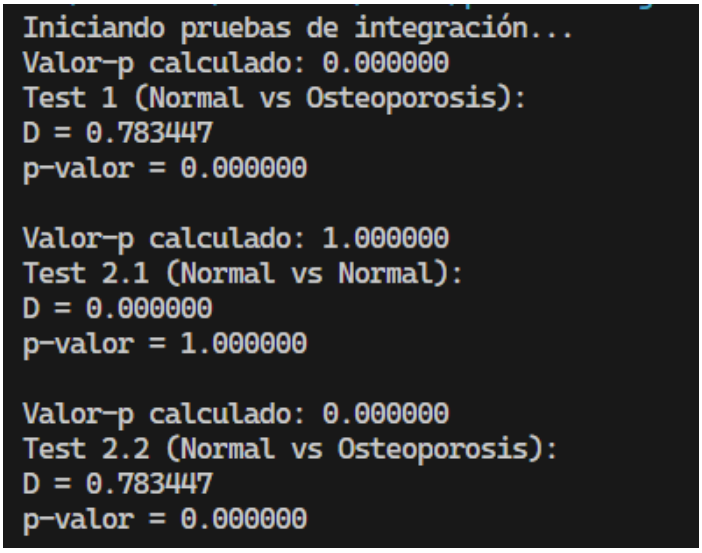
Resultados Esperados

- P-valor ≈ 0 para tejidos diferentes (normal vs osteoporosis)
- P-valor ≈ 1 para tejidos idénticos (normal vs normal)
- Estadístico D menor para tejidos similares que para diferentes

```
def test_integracion_completa():
    """Prueba de integración del sistema completo"""

    print("Iniciando pruebas de integración...")

    # Generar datos de prueba
    tipos_tejido = [TipoTejido.NORMAL, TipoTejido.OSTEOPENIA,
TipoTejido.OSTEOPOROSIS]
    imagenes_test = {}

    for tipo in tipos_tejido:
        imagenes_test[tipo] = generar_imagen_hueso_simulada(
            width=64, height=64,
            tipo=tipo,
            seed=42
        )

    # Test 1: Flujo completo de análisis
    resultado = prueba_ks_imagenes(
        imagenes_test[TipoTejido.NORMAL],
        imagenes_test[TipoTejido.OSTEOPOROSIS]
    )

    assert 'estadistico_D' in resultado
    assert 'p_valor' in resultado
    assert 'conclusion' in resultado

    # Test 2: Validar coherencia de resultados
    # Normal vs Normal debería ser similar
    resultado_similar = prueba_ks_imagenes(
        imagenes_test[TipoTejido.NORMAL],
        imagenes_test[TipoTejido.NORMAL]
    )

    # Normal vs Osteoporosis debería ser diferente
    resultado_diferente = prueba_ks_imagenes(
        imagenes_test[TipoTejido.NORMAL],
        imagenes_test[TipoTejido.OSTEOPOROSIS]
    )
```

```
    assert resultado_diferente['estadistico_D'] > resultado_similar['estadistico_D']

    print("✓ Pruebas de integración completadas exitosamente")

test_integracion_completa()
```

Output:



Pruebas de rendimiento

Descripción del Test

El test de rendimiento evalúa el comportamiento del sistema en términos de:

1. **Tiempo de Procesamiento**
   - Generación de imágenes
   - Análisis KS
2. **Uso de Memoria**
   - Consumo de memoria durante la generación
   - Consumo durante el análisis
3. **Escalabilidad** Prueba diferentes tamaños de imagen:
   - 64x64 (Pequeña)
   - 128x128 (Mediana)
   - 256x256 (Mediana - standard)
   - 512x512 (Grande)
   - 2000x2000 (Muy grande)

```
def test_rendimiento():
  """Evaluación de rendimiento del sistema"""

  tamaños = [(64, 64), (128, 128), (256, 256), (512, 512), (2000, 2000)]
  resultados_rendimiento = []

  for width, height in tamaños:
    print(f"Probando rendimiento para {width}x{height}...")

    # Medir tiempo de generación
    start_time = time.time()
    img1 = generar_imagen_hueso_simulada(width=width, height=height, seed=1)
    img2 = generar_imagen_hueso_simulada(width=width, height=height, seed=2)
    tiempo_generacion = time.time() - start_time

    # Medir memoria antes del análisis
    proceso = psutil.Process(os.getpid())
    memoria_inicial = proceso.memory_info().rss / 1024 / 1024  # MB

    # Medir tiempo de análisis
    start_time = time.time()
    resultado = prueba_ks_imagenes(img1, img2)
    tiempo_analisis = time.time() - start_time

    # Medir memoria después del análisis
    memoria_final = proceso.memory_info().rss / 1024 / 1024  # MB
```

```
        uso_memoria = memoria_final - memoria_inicial

        resultados_rendimiento.append({
            'dimensiones': f"{width}x{height}",
            'pixeles': width * height,
            'tiempo_generacion': tiempo_generacion,
            'tiempo_analisis': tiempo_analisis,
            'uso_memoria_mb': uso_memoria
        })

    # Mostrar resultados
    print("\nRESULTADOS DE RENDIMIENTO:")
    print("-" * 70)
    print(f"{'Dimensiones':<12}{'Píxeles':<8}{'Gen(s)':<8}{'Análisis(s)':<12}{'Memoria(MB)':<12}")
    print("-" * 70)

    for r in resultados_rendimiento:
        print(f"{r['dimensiones']:<12}{r['pixeles']:<8} "
              f"{r['tiempo_generacion']:<8.3f}{r['tiempo_analisis']:<12.3f} "
              f"{r['uso_memoria_mb']:<12.1f}")

    return resultados_rendimiento

# Ejecutar pruebas de rendimiento
resultados_perf = test_rendimiento()
```

Output:

```
Probando rendimiento para 64x64...
Valor-p calculado: 0.732979
Probando rendimiento para 128x128...
Valor-p calculado: 0.000005
Probando rendimiento para 256x256...
Valor-p calculado: 0.792734
Probando rendimiento para 512x512...
Valor-p calculado: 0.004491
Probando rendimiento para 2000x2000...
Valor-p calculado: 0.000000

RESULTADOS DE RENDIMIENTO:
----------------------------------------------------------------------
Dimensiones  Píxeles  Gen(s)   Análisis(s)  Memoria(MB)
----------------------------------------------------------------------
64x64        4096     0.043    0.001        0.1
128x128      16384    0.015    0.001        0.2
256x256      65536    0.056    0.004        -0.4
512x512      262144   0.242    0.013        0.3
2000x2000    4000000  3.937    0.224        -0.1
```

ANÁLISIS DE RENDIMIENTO

Métricas de rendimiento

Complejidad Temporal

- **Generación de imágenes**: O(n×m) donde n×m son las dimensiones
- **Cálculo de histograma**: O(n×m)
- **Análisis KS**: O(k) donde k=256 (niveles de gris)
- **Complejidad total**: O(n×m + k) ≈ O(n×m)

Optimizaciones implementadas

1. Vectorización NumPy: Uso de operaciones vectorizadas para cálculos masivos
2. Algoritmo KS Optimizado: Implementación eficiente de la serie infinita
3. Gestión de Memoria: Liberación proactiva de arrays grandes
4. Cacheo de Parámetros: Precálculo de parámetros de tejido

Benchmarks

Propósito
Este módulo evalúa el rendimiento del sistema midiendo:

1. Métricas de Velocidad:

- Tiempo de generación de imágenes (256x256)
- Tiempo de análisis KS
- Tiempo total de procesamiento
- Rendimiento del sistema (análisis por segundo)

2. Metodología de Prueba:

- Utiliza el módulo `timeit` para mediciones precisas
- Ejecuta cada operación 10 veces y promedia los resultados
- Prueba con tamaño de imagen constante (256x256)
- Usa diferentes semillas para las imágenes de prueba

```
def benchmark_completo():
  """Prueba de rendimiento completa del sistema"""

  # Benchmark generación
  tiempo_gen = timeit.timeit(
    lambda: generar_imagen_hueso_simulada(256, 256),
    number=10
  ) / 10

  # Benchmark análisis KS
  img1 = generar_imagen_hueso_simulada(256, 256, seed=1)
  img2 = generar_imagen_hueso_simulada(256, 256, seed=2)

  tiempo_ks = timeit.timeit(
    lambda: prueba_ks_imagenes(img1, img2),
    number=10
  ) / 10
```

```
    print("BENCHMARKS:")
    print(f"Generación (256x256): {tiempo_gen:.4f}s")
    print(f"Análisis KS (256x256): {tiempo_ks:.4f}s")
    print(f"Tiempo total: {tiempo_gen + tiempo_ks:.4f}s")

    # Throughput
    imagenes_por_segundo = 1 / (tiempo_gen + tiempo_ks)
    print(f"Throughput: {imagenes_por_segundo:.1f} análisis/segundo")

benchmark_completo()
```

Output:

```
Valor-p calculado: 0.792734
Valor-p calculado: 0.792734
Valor-p calculado: 0.792734
Valor-p calculado: 0.792734
Valor-p calculado: 0.792734
Valor-p calculado: 0.792734
Valor-p calculado: 0.792734
Valor-p calculado: 0.792734
Valor-p calculado: 0.792734
Valor-p calculado: 0.792734
BENCHMARKS:
Generación (256x256): 0.0291s
Análisis KS (256x256): 0.0053s
Tiempo total: 0.0344s
Throughput: 29.1 análisis/segundo
```

SOLUCIÓN DE PROBLEMAS

Errores comunes

Tipo de error	Posible causa	Solución recomendada
FileNotFoundError	Ruta incorrecta o archivo no cargado	Verifica que el archivo esté en la ubicación correcta y que se haya seleccionado correctamente en la interfaz.
ValueError: operands could not be broadcast	Imágenes con dimensiones diferentes	Asegúrate de que las imágenes a comparar tengan la misma resolución o estén preprocesadas adecuadamente.
TypeError: unsupported operand type(s)	Tipos de datos inconsistentes en operaciones numéricas	Asegúrate de que las entradas sean válidas y convertidas correctamente (por ejemplo, usando float).
tk.TclError	Problemas con la GUI (Tkinter) en entornos no gráficos	Ejecuta el programa en un entorno con soporte gráfico, como una máquina local o entorno virtual con GUI.

Errores de entrada

- **Entrada nula o no seleccionada**:
  Si no se carga una imagen antes de ejecutar funciones como el análisis KS o la simulación, el sistema puede lanzar excepciones o comportarse de forma inesperada.

  Solución: Agregar validaciones previas al análisis para confirmar que se hayan cargado correctamente las imágenes.

- **Formatos de imagen no soportados**:
  Aunque se usa Pillow y OpenCV, algunos formatos poco comunes pueden fallar.

  Solución: Convertir las imágenes a formatos estándar como .png o .jpg.

- **Archivos dañados o corruptos**:
  Si una imagen no puede ser leída, OpenCV puede lanzar un error silencioso y devolver None.

  Solución: Validar explícitamente si la imagen fue cargada (if img is None:) antes de continuar.

Errores de memoria

- **Imágenes demasiado grandes**:
  Al usar imágenes de alta resolución (>3000x3000 px), los cálculos de comparación pueden requerir una gran cantidad de memoria.

  Solución: Escalar imágenes a una resolución manejable antes del análisis. Utilizar herramientas como psutil para monitorear el uso de RAM.

- **Procesamiento de grandes volúmenes en dataset**:
  Cuando se generan múltiples imágenes de simulación (por ejemplo, >100), puede saturarse la memoria temporal.

  Solución: Limitar la cantidad de imágenes simuladas por sesión o utilizar procesamiento por lotes.

Errores de convergencia

- **$\lambda \leq 0$ en el test KS**:
  Este valor implica que las imágenes son estadísticamente idénticas. Aunque esto no es un error, puede causar confusión.

  Solución: Mostrar un mensaje claro al usuario indicando que no se encontraron diferencias significativas.

- **Serie infinita no converge lo suficiente**:
  En valores de λ muy grandes, la suma infinita puede necesitar muchos términos para aproximarse correctamente al valor-p.

  Solución: Usar un límite razonable de términos (max_terminos) y un umbral de convergencia como 1e-10.

- **Comparaciones incorrectas por escalado distinto**:
  Las imágenes que visualmente parecen iguales pueden fallar en el análisis KS si tienen histogramas afectados por contraste o brillo.

  Solución: Normalizar histograma antes del análisis (opcional).

FAQ (Preguntas frecuentes)

P: ¿Por qué el análisis KS rechaza H0 con imágenes aparentemente similares?

R: Esto puede deberse a diferencias sutiles en la distribución de intensidades. Usar debug_analisis_ks() para investigar.

P: ¿Cómo mejorar el rendimiento con imágenes muy grandes?

R: Considerar:

1. Reducir el tamaño de imagen manteniendo la información relevante
2. Usar muestreo aleatorio de píxeles
3. Implementar procesamiento en paralelo

P: ¿El método funciona con imágenes reales de rayos X?

R: El sistema está diseñado para imágenes simuladas. Para imágenes reales, se requiere:

1. Preprocesamiento específico
2. Normalización de intensidades
3. Calibración de parámetros

MANTENIMIENTO Y EXTENSIONES

Guía de mantenimiento

- Revisar requirements.txt periódicamente.
- Comprobar compatibilidad con nuevas versiones de Python (sugerido: Python 3.9 o superior).
- Documentar funciones nuevas y mantener el estilo PEP8.

Extensiones posibles

- Comparación de imágenes en color usando distribución multicanal.
- Aplicación con radiografías reales.
- Estudio de sets con Machin Learning.
- Versión web con mejores interfaces y presentado como servicio.

Actualizaciones futuras

- Agregar análisis por lotes y exportación automática de reportes.
- Soporte para bases de datos de imágenes clínicas reales.

CONCLUSION

Este proyecto representa una convergencia significativa entre estadística computacional, visión por computadora e interfaces gráficas interactivas, aplicadas al análisis de imágenes médicas simuladas para el diagnóstico de condiciones como la osteoporosis. A través de la implementación del test de Kolmogórov-Smirnov, se logró construir una herramienta capaz de identificar diferencias estadísticas relevantes entre estructuras óseas simuladas, abriendo camino a aplicaciones futuras en entornos clínicos y educativos.

La metodología empleada, que combina simulación controlada de tejidos, análisis estadístico riguroso y visualización gráfica intuitiva, demuestra que es posible evaluar la distribución de patrones internos de imágenes con un enfoque no paramétrico, robusto y sensible. Particularmente, el valor de λ y su correspondiente valor-p permiten interpretar cuantitativamente las similitudes o discrepancias entre muestras, incluso en presencia de ruido u otras irregularidades visuales.

Desde el punto de vista técnico, se puso especial atención en la modularidad del código, la escalabilidad de las funciones y la capacidad de extender el sistema para nuevos tipos de análisis. Asimismo, se contemplaron casos especiales que podrían comprometer la validez de los resultados, como la convergencia del valor-p cuando λ tiende a cero (caso de imágenes idénticas) o el tratamiento de errores por archivos corruptos o entradas inválidas.

Este sistema no solo permite realizar análisis automatizados, sino que también sirve como una plataforma educativa para comprender la sensibilidad del test KS en contextos visuales. Además, su implementación en Python garantiza portabilidad, acceso abierto y capacidad de integración con otros sistemas de análisis o inteligencia artificial.

En conclusión, el proyecto no es solo una solución técnica puntual, sino un ejemplo de cómo la estadística avanzada puede integrarse de forma efectiva con tecnologías visuales para generar conocimiento útil, reproducible y extensible en campos tan relevantes como la medicina, la ingeniería biomédica y el aprendizaje automático.

ANEXOS

Código fuente completo

Ks.py

```
import numpy as np
import cv2

def calcular_histograma_normalizado(imagen: np.ndarray) -> np.ndarray:
    """
    Calcula el histograma normalizado de una imagen en escala de grises.

    Args:
        imagen: Array numpy de la imagen en escala de grises

    Returns:
        Array con el histograma normalizado (probabilidades)
    """
    # Calcular histograma para niveles 0-255
    hist, _ = np.histogram(imagen.flatten(), bins=256, range=(0, 256))

    # Normalizar para obtener probabilidades
    hist_normalizado = hist / np.sum(hist)

    return hist_normalizado

def calcular_fda_empirica(histograma_normalizado: np.ndarray) -> np.ndarray:
    """
    Calcula la función de distribución acumulada (FDA) empírica.

    Args:
        histograma_normalizado: Histograma normalizado de la imagen

    Returns:
        Array con la FDA empírica F_g
    """
    # F_g = sum(h_g' para g'=0 hasta g)
    fda = np.cumsum(histograma_normalizado)

    return fda
```

```
def calcular_estadistico_ks(fda1: np.ndarray, fda2: np.ndarray) -> float:
    """
    Calcula el estadístico D^ de Kolmogorov-Smirnov.

    Args:
        fda1: FDA de la primera imagen
        fda2: FDA de la segunda imagen

    Returns:
        Valor del estadístico D^
    """
    # D^ = max_g |F_g^(1) - F_g^(2)|
    diferencias = np.abs(fda1 - fda2)
    d_hat = np.max(diferencias)

    return d_hat

def calcular_lambda_ks(d_hat: float, p1: int, q1: int, p2: int, q2: int) -> float:
    """
    Calcula el parámetro λ_KS para la prueba KS.

    Args:
        d_hat: Estadístico D^ calculado
        p1, q1: Dimensiones de la primera imagen
        p2, q2: Dimensiones de la segunda imagen

    Returns:
        Valor de λ_KS
    """
    # J = (P1*Q1 * P2*Q2) / (P1*Q1 + P2*Q2)
    n1 = p1 * q1  # Número total de píxeles imagen 1
    n2 = p2 * q2  # Número total de píxeles imagen 2

    j = (n1 * n2) / (n1 + n2)

    # λ_KS = D^ * sqrt(J + 0.11/sqrt(J + 0.12))
    lambda_ks = d_hat * (np.sqrt(j) + (0.11 / np.sqrt(j)) + 0.12)

    return lambda_ks

def calcular_q_ks(lambda_val: float, max_terminos: int = 100) -> float:
```

```
    """
    Calcula el valor-p usando la función Q_KS.

    Args:
        lambda_val: Valor de λ_KS
        max_terminos: Número máximo de términos en la serie infinita

    Returns:
        Valor de Q_KS (valor-p)
    """
    # Q_KS(λ) = 2 * sum(j=1 to inf) (-1)^(j-1) * exp(-2*j^2*λ^2)
    q_ks = 0.0

    if lambda_val <= 0:
        return 1.0

    for j in range(1, max_terminos + 1):
        termino = ((-1) ** (j - 1)) * np.exp(-2 * j**2 * lambda_val**2)
        q_ks += termino

        # Criterio de convergencia para series infinitas
        if abs(termino) < 1e-10:
            break

    q_ks *= 2

    # Asegurar que esté en rango [0, 1]
    q_ks = max(0.0, min(1.0, q_ks))

    return q_ks

def prueba_ks_imagenes(imagen1: np.ndarray, imagen2: np.ndarray,
                       alpha: float = 0.05) -> dict:
    """
    Realiza la prueba completa de Kolmogorov-Smirnov entre dos imágenes.

    Args:
        imagen1: Primera imagen en escala de grises
        imagen2: Segunda imagen en escala de grises
        alpha: Nivel de significancia (por defecto 0.05)

    Returns:
```

```
    Diccionario con los resultados de la prueba
"""
# Convertir a escala de grises si es necesario
if len(imagen1.shape) == 3:
    imagen1 = cv2.cvtColor(imagen1, cv2.COLOR_BGR2GRAY)
if len(imagen2.shape) == 3:
    imagen2 = cv2.cvtColor(imagen2, cv2.COLOR_BGR2GRAY)

# Paso 1: Calcular histogramas normalizados
hist1 = calcular_histograma_normalizado(imagen1)
hist2 = calcular_histograma_normalizado(imagen2)

# Paso 2: Calcular FDA empíricas
fda1 = calcular_fda_empirica(hist1)
fda2 = calcular_fda_empirica(hist2)

# Paso 3: Calcular estadístico D^
d_hat = calcular_estadistico_ks(fda1, fda2)

# Paso 4: Calcular λ_KS
p1, q1 = imagen1.shape
p2, q2 = imagen2.shape
lambda_ks = calcular_lambda_ks(d_hat, p1, q1, p2, q2)

# Paso 5: Calcular valor-p (Q_KS)
p_valor = calcular_q_ks(lambda_ks)
print(f"Valor-p calculado: {p_valor:.6f}")

# Paso 6: Decisión estadística
rechazar_h0 = p_valor < alpha

# Preparar resultados
resultados = {
    'estadistico_D': d_hat,
    'lambda_KS': lambda_ks,
    'p_valor': p_valor,
    'alpha': alpha,
    'rechazar_H0': rechazar_h0,
    'conclusion': 'Las imágenes tienen distribuciones diferentes' if rechazar_h0
                  else 'No hay evidencia suficiente de diferencia entre distribuciones',
    'histograma_1': hist1,
    'histograma_2': hist2,
```

```
        'fda_1': fda1,
        'fda_2': fda2
    }

    return resultados

# Ejemplo de uso
def ejemplo_uso():
    """
    Ejemplo de cómo usar la función de prueba KS.
    """
    # Simular dos imágenes diferentes
    np.random.seed(42)

    # Imagen 1: distribución normal
    imagen1 = np.random.normal(128, 40, (100, 100)).astype(np.uint8)
    imagen1 = np.clip(imagen1, 0, 255)

    # Imagen 2: distribución uniforme
    imagen2 = np.random.uniform(0, 255, (100, 100)).astype(np.uint8)

    # Realizar prueba KS
    resultado = prueba_ks_imagenes(imagen1, imagen2)

    # Mostrar resultados
    print("=== RESULTADOS PRUEBA KOLMOGOROV-SMIRNOV ===")
    print(f"Estadístico D^: {resultado['estadistico_D']:.6f}")
    print(f"Lambda KS: {resultado['lambda_KS']:.6f}")
    print(f"Valor-p: {resultado['p_valor']:.6f}")
    print(f"Nivel de significancia: {resultado['alpha']}")
    print(f"Rechazar H0: {resultado['rechazar_H0']}")
    print(f"Conclusión: {resultado['conclusion']}")

if __name__ == "__main__":
    ejemplo_uso()
```

generadorTejidos.py

```
import numpy as np
from typing import Tuple, Dict, Optional
from enum import Enum

class TipoTejido(Enum):
  """Enumeration para los tipos de tejido óseo."""
  NORMAL = "normal"
  OSTEOPENIA = "osteopenia"
  OSTEOPOROSIS = "osteoporosis"

class ParametrosTejido:
  """Clase para almacenar parámetros específicos de cada tipo de tejido."""

  def __init__(self, densidad_base: float, rugosidad: float,
         conectividad: float, porosidad: float,
         intensidad_cortical: Tuple[int, int],
         intensidad_trabecular: Tuple[int, int]):
    """
    Args:
      densidad_base: Densidad base del tejido (0.0-1.0)
      rugosidad: Nivel de rugosidad de la textura (0.0-1.0)
      conectividad: Conectividad trabecular (0.0-1.0)
      porosidad: Nivel de porosidad/espacios vacíos (0.0-1.0)
      intensidad_cortical: Rango de intensidades para hueso cortical (min, max)
      intensidad_trabecular: Rango de intensidades para hueso trabecular (min, max)
    """
    self.densidad_base = densidad_base
    self.rugosidad = rugosidad
    self.conectividad = conectividad
    self.porosidad = porosidad
    self.intensidad_cortical = intensidad_cortical
    self.intensidad_trabecular = intensidad_trabecular

def obtener_parametros_tejido(tipo: TipoTejido) -> ParametrosTejido:
  """
  Retorna los parámetros específicos para cada tipo de tejido.
  """
  parametros = {
    TipoTejido.NORMAL: ParametrosTejido(
      densidad_base=0.9,
```

```
            rugosidad=0.2,
            conectividad=0.9,
            porosidad=0.15,  # Menos poros
            intensidad_cortical=(180, 220),
            intensidad_trabecular=(120, 160)
        ),

        TipoTejido.OSTEOPENIA: ParametrosTejido(
            densidad_base=0.7,
            rugosidad=0.4,
            conectividad=0.7,
            porosidad=0.35,  # Poros intermedios
            intensidad_cortical=(150, 190),
            intensidad_trabecular=(90, 130)
        ),

        TipoTejido.OSTEOPOROSIS: ParametrosTejido(
            densidad_base=0.5,
            rugosidad=0.6,
            conectividad=0.4,
            porosidad=0.55,  # Más poros y más grandes
            intensidad_cortical=(120, 160),
            intensidad_trabecular=(60, 100)
        )
    }

    return parametros[tipo]

def generar_ruido_perlin_2d(width: int, height: int, scale: float = 0.1,
                            octaves: int = 4, persistence: float = 0.5,
                            lacunarity: float = 2.0, seed: Optional[int] = None) -> np.ndarray:
    """
    Genera ruido Perlin 2D simplificado para simular textura trabecular.

    Args:
        width, height: Dimensiones de la imagen
        scale: Escala del ruido (más pequeño = más detalle)
        octaves: Número de octavas (capas de detalle)
        persistence: Amplitud relativa entre octavas
        lacunarity: Frecuencia relativa entre octavas
        seed: Semilla para reproducibilidad
```

```
    Returns:
        Array 2D con valores de ruido normalizados [0, 1]
    """
    if seed is not None:
        np.random.seed(seed)

    # Implementación simplificada de ruido Perlin usando interpolación
    noise = np.zeros((height, width))

    for octave in range(octaves):
        # Frecuencia y amplitud para esta octava
        freq = scale * (lacunarity ** octave)
        amp = persistence ** octave

        # Generar ruido base para esta octava
        x_coords = np.arange(width) * freq
        y_coords = np.arange(height) * freq

        # Crear grid de coordenadas
        xx, yy = np.meshgrid(x_coords, y_coords)

        # Generar ruido usando funciones trigonométricas
        octave_noise = (np.sin(xx) * np.cos(yy) +
                        np.sin(xx * 2.1) * np.cos(yy * 1.7) +
                        np.sin(xx * 0.8) * np.cos(yy * 2.3)) * amp

        noise += octave_noise

    # Normalizar a [0, 1]
    noise = (noise - noise.min()) / (noise.max() - noise.min())

    return noise

def generar_patron_trabecular(width: int, height: int,
                              parametros: ParametrosTejido,
                              seed: Optional[int] = None) -> np.ndarray:
    """
    Genera un patrón trabecular con poros circulares no superpuestos.
    """
    if seed is not None:
        np.random.seed(seed)
```

```
    # Calcular tamaño máximo de poro basado en dimensiones
    dimension_min = min(width, height)

    # Ajustar tamaños para dimensiones pequeñas
    if dimension_min < 50:
        tamano_max_poro = max(3, int(dimension_min * 0.2))  # 20% para imágenes pequeñas
        tamano_min_poro = max(2, int(dimension_min * 0.1))  # 10% para imágenes pequeñas
    else:
        tamano_max_poro = int(dimension_min * 0.15)  # 15% de la dimensión menor
        tamano_min_poro = max(3, int(dimension_min * 0.02))  # 2% de la dimensión menor

    # Asegurar que min < max
    if tamano_min_poro >= tamano_max_poro:
        tamano_min_poro = max(2, tamano_max_poro - 1)

    # Resto del código igual...
    patron = np.ones((height, width))
    poros_existentes = []

    # Ajustar área objetivo para imágenes pequeñas
    area_total = width * height
     factor_area = min(1.0, dimension_min / 100.0)  # Reducir área objetivo para imágenes
pequeñas
    area_objetivo = area_total * parametros.porosidad * factor_area
    area_actual = 0
    max_intentos = 1000
    intentos = 0

    # Crear grid de coordenadas para cálculos
    y, x = np.ogrid[:height, :width]

    while area_actual < area_objetivo and intentos < max_intentos:
        # Usar uniform en lugar de randint para más control
        radio = np.random.uniform(tamano_min_poro, tamano_max_poro)
        radio = int(radio)

        # Asegurar que el radio es válido
        if radio < 2:
            radio = 2

        # Asegurar que el centro es válido
        margen = radio + 1
```

```
        if width <= 2*margen or height <= 2*margen:
            break

        cx = np.random.randint(margen, width - margen)
        cy = np.random.randint(margen, height - margen)

        # Verificar superposición con poros existentes
        superpuesto = False
        for px, py, pr in poros_existentes:
            distancia = np.sqrt((cx - px)**2 + (cy - py)**2)
            if distancia < (radio + pr + 2):  # +2 para dejar espacio entre poros
                superpuesto = True
                break

        if not superpuesto:
            # Crear y aplicar el nuevo poro
            distancia = np.sqrt((x - cx)**2 + (y - cy)**2)
            mascara = distancia <= radio

            # Crear borde suave
            factor_suavizado = np.clip(1 - (distancia / radio), 0, 1)
            patron[mascara] *= factor_suavizado[mascara]

            # Registrar el poro
            poros_existentes.append((cx, cy, radio))
            area_actual += np.pi * radio**2

        intentos += 1

    # Ajustar densidad según conectividad
    patron = patron * (parametros.conectividad * 0.5 + 0.5)

    # Aplicar ruido sutil para textura
    ruido = generar_ruido_perlin_2d(
        width, height,
        scale=0.15,
        octaves=2,
        persistence=0.3,
        lacunarity=2.0,
        seed=seed
    )
```

```
    # Combinar patrón con ruido de manera más sutil
    patron = patron * (0.95 + 0.05 * ruido)

    # Normalizar
    patron = (patron - patron.min()) / (patron.max() - patron.min())

    return patron

def generar_imagen_hueso_simulada(width: int = 256, height: int = 256,
                                  tipo: TipoTejido = TipoTejido.NORMAL,
                                  incluir_cortical: bool = True,
                                  grosor_cortical: int = 15,
                                  seed: Optional[int] = None) -> np.ndarray:
    """
    Genera una imagen simulada de tejido óseo con características específicas.

    Args:
        width, height: Dimensiones de la imagen
        tipo: Tipo de tejido óseo a simular
        incluir_cortical: Si incluir hueso cortical en los bordes
        grosor_cortical: Grosor del hueso cortical en píxeles
        seed: Semilla para reproducibilidad

    Returns:
        Array 2D con la imagen simulada en escala de grises [0, 255]
    """
    if seed is not None:
        np.random.seed(seed)

    # Obtener parámetros para el tipo de tejido
    parametros = obtener_parametros_tejido(tipo)

    # Generar patrón trabecular
    patron_trabecular = generar_patron_trabecular(width, height, parametros, seed)

    # Convertir a intensidades de escala de grises
    min_int, max_int = parametros.intensidad_trabecular
    imagen = (patron_trabecular * (max_int - min_int) + min_int).astype(np.uint8)

    # Añadir hueso cortical si se requiere
    if incluir_cortical:
        # Crear máscara para el hueso cortical (bordes)
```

```
        cortical_mask = np.zeros((height, width), dtype=bool)

        # Bordes externos
        cortical_mask[:grosor_cortical, :] = True  # Superior
        cortical_mask[-grosor_cortical:, :] = True  # Inferior
        cortical_mask[:, :grosor_cortical] = True  # Izquierdo
        cortical_mask[:, -grosor_cortical:] = True  # Derecho

        # Generar intensidades para hueso cortical
        min_cort, max_cort = parametros.intensidad_cortical
        intensidad_cortical = np.random.randint(min_cort, max_cort + 1,
                                                size=cortical_mask.sum())

        # Aplicar hueso cortical
        imagen[cortical_mask] = intensidad_cortical

    # Añadir ruido realista
    ruido = np.random.normal(0, 5, (height, width))
    imagen = np.clip(imagen.astype(float) + ruido, 0, 255).astype(np.uint8)

    return imagen

def generar_dataset_simulado(n_imagenes_por_tipo: int = 50,
                             dimensiones: Tuple[int, int] = (256, 256),
                             incluir_cortical: bool = True,
                             seed_base: Optional[int] = None) -> Dict[str, np.ndarray]:
    """
    Genera un dataset completo de imágenes simuladas para los tres tipos de tejido.

    Args:
        n_imagenes_por_tipo: Número de imágenes a generar por cada tipo
        dimensiones: Tupla (width, height) para el tamaño de las imágenes
        incluir_cortical: Si incluir hueso cortical
        seed_base: Semilla base para reproducibilidad

    Returns:
        Diccionario con arrays de imágenes por tipo
    """
    dataset = {}
    width, height = dimensiones

    for tipo in TipoTejido:
```

```
        print(f"Generando {n_imagenes_por_tipo} imágenes de tipo {tipo.value}...")

        imagenes = []
        for i in range(n_imagenes_por_tipo):
            # Usar semilla diferente para cada imagen
            semilla = (seed_base + i) if seed_base is not None else None

            imagen = generar_imagen_hueso_simulada(
                width=width,
                height=height,
                tipo=tipo,
                incluir_cortical=incluir_cortical,
                seed=semilla+i
            )

            imagenes.append(imagen)

        dataset[tipo.value] = np.array(imagenes)

    return dataset

def visualizar_comparacion_tipos(seed: int = 42) -> None:
    """
    Genera y muestra ejemplos de los tres tipos de tejido para comparación.
    """
    import matplotlib.pyplot as plt

    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    for i, tipo in enumerate(TipoTejido):
        imagen = generar_imagen_hueso_simulada(
            width=256, height=256,
            tipo=tipo,
            seed=seed + i
        )

        axes[i].imshow(imagen, cmap='gray', vmin=0, vmax=255)
        axes[i].set_title(f'Tejido {tipo.value.capitalize()}')
        axes[i].axis('off')

    plt.tight_layout()
    plt.show()
```

```
# Ejemplo de uso
def ejemplo_uso():
    """Ejemplo de cómo usar el simulador."""

    print("=== SIMULADOR DE IMÁGENES DE OSTEOPOROSIS ===\n")

    # Generar una imagen de cada tipo
    for tipo in TipoTejido:
        imagen = generar_imagen_hueso_simulada(
            width=128, height=128,
            tipo=tipo,
            seed=42
        )

        print(f"Imagen {tipo.value}:")
        print(f"  - Dimensiones: {imagen.shape}")
        print(f"  - Rango intensidades: [{imagen.min()}, {imagen.max()}]")
        print(f"  - Intensidad promedio: {imagen.mean():.1f}")
        print()

    # Generar dataset pequeño
    print("Generando dataset de prueba...")
    dataset = generar_dataset_simulado(
        n_imagenes_por_tipo=5,
        dimensiones=(64, 64),
        seed_base=123
    )

    for tipo, imagenes in dataset.items():
        print(f"Dataset {tipo}: {imagenes.shape}")

if __name__ == "__main__":
    ejemplo_uso()
```

ui.py

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import numpy as np
import cv2
from PIL import Image, ImageTk
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import ks
import generadorTejidos as gt

# Importar las funciones que ya creamos (simulador y KS)
# (Aquí incluirías los imports de tus módulos anteriores)


class DetectorOsteoporosisApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Detector de Osteoporosis - Análisis Kolmogorov-Smirnov")
        self.root.geometry("1000x700")
        self.root.configure(bg='#f0f0f0')

        # Variables para almacenar imágenes
        self.imagen1 = None
        self.imagen2 = None
        self.resultados_ks = None

        self.crear_interfaz()

    def crear_interfaz(self):
        """Crea la interfaz principal con pestañas."""
        # Título principal
        titulo = tk.Label(self.root, text="Detector de Osteoporosis",
                          font=("Arial", 20, "bold"), bg='#f0f0f0', fg='#333')
        titulo.pack(pady=10)

        subtitulo = tk.Label(self.root, text="Análisis mediante Prueba Kolmogorov-Smirnov",
                             font=("Arial", 12), bg='#f0f0f0', fg='#666')
        subtitulo.pack(pady=(0, 20))
```

```
    # Crear notebook para pestañas
    self.notebook = ttk.Notebook(self.root)
    self.notebook.pack(fill='both', expand=True, padx=20, pady=10)

    # Crear pestañas
    self.crear_pestana_simulaciones()
    self.crear_pestana_analisis_ks()

def crear_pestana_simulaciones(self):
    """Crea la pestaña de simulaciones."""
    frame_sim = ttk.Frame(self.notebook)
    self.notebook.add(frame_sim, text="Generador de Simulaciones")

    # Frame para controles
    frame_controles = tk.Frame(frame_sim, bg='white', relief='raised', bd=2)
    frame_controles.pack(fill='x', padx=10, pady=10)

    tk.Label(frame_controles, text="Parámetros de Simulación",
          font=("Arial", 14, "bold"), bg='white').pack(pady=10)

    # Tipo de tejido
    tk.Label(frame_controles, text="Tipo de Tejido:", bg='white').pack(anchor='w', padx=20)
    self.tipo_tejido = tk.StringVar(value="normal")
    frame_tipo = tk.Frame(frame_controles, bg='white')
    frame_tipo.pack(fill='x', padx=20, pady=5)

    tk.Radiobutton(frame_tipo, text="Normal", variable=self.tipo_tejido,
              value="normal", bg='white').pack(side='left', padx=10)
    tk.Radiobutton(frame_tipo, text="Osteopenia", variable=self.tipo_tejido,
              value="osteopenia", bg='white').pack(side='left', padx=10)
    tk.Radiobutton(frame_tipo, text="Osteoporosis", variable=self.tipo_tejido,
              value="osteoporosis", bg='white').pack(side='left', padx=10)

    # Dimensiones
    frame_dim = tk.Frame(frame_controles, bg='white')
    frame_dim.pack(fill='x', padx=20, pady=10)

    tk.Label(frame_dim, text="Ancho:", bg='white').pack(side='left')
    self.ancho_var = tk.StringVar(value="256")
    tk.Entry(frame_dim, textvariable=self.ancho_var, width=8).pack(side='left', padx=5)

    tk.Label(frame_dim, text="Alto:", bg='white').pack(side='left', padx=(20,0))
```

```
        self.alto_var = tk.StringVar(value="256")
        tk.Entry(frame_dim, textvariable=self.alto_var, width=8).pack(side='left', padx=5)

        # Checkboxes
        self.incluir_cortical = tk.BooleanVar(value=True)
        tk.Checkbutton(frame_controles, text="Incluir hueso cortical",
                       variable=self.incluir_cortical, bg='white').pack(anchor='w', padx=20, pady=5)

        # Semilla
        frame_semilla = tk.Frame(frame_controles, bg='white')
        frame_semilla.pack(fill='x', padx=20, pady=5)
        tk.Label(frame_semilla, text="Semilla (opcional):", bg='white').pack(side='left')
        self.semilla_var = tk.StringVar(value="42")
        tk.Entry(frame_semilla, textvariable=self.semilla_var, width=10).pack(side='left', padx=5)

        # Botones
        frame_botones = tk.Frame(frame_controles, bg='white')
        frame_botones.pack(fill='x', padx=20, pady=15)

        tk.Button(frame_botones, text="Generar Imagen Individual",
                  command=self.generar_imagen_individual, bg='#4CAF50', fg='white',
                  font=("Arial", 10, "bold")).pack(side='left', padx=5)

        tk.Button(frame_botones, text="Generar Dataset",
                  command=self.generar_dataset, bg='#2196F3', fg='white',
                  font=("Arial", 10, "bold")).pack(side='left', padx=5)

        tk.Button(frame_botones, text="Comparar Tipos",
                  command=self.comparar_tipos, bg='#FF9800', fg='white',
                  font=("Arial", 10, "bold")).pack(side='left', padx=5)

        # Área de visualización
        self.frame_visualizacion = tk.Frame(frame_sim, bg='white', relief='sunken', bd=2)
        self.frame_visualizacion.pack(fill='both', expand=True, padx=10, pady=10)

        self.label_imagen_sim = tk.Label(self.frame_visualizacion, text="Las imágenes generadas
aparecerán aquí",
                                         bg='white', fg='#666')
        self.label_imagen_sim.pack(expand=True)

    def crear_pestana_analisis_ks(self):
        """Crea la pestaña de análisis KS."""
```

```
        frame_ks = ttk.Frame(self.notebook)
        self.notebook.add(frame_ks, text="Análisis Kolmogorov-Smirnov")

        # Frame superior para selección de imágenes
        frame_superior = tk.Frame(frame_ks, bg='white', relief='raised', bd=2)
        frame_superior.pack(fill='x', padx=10, pady=10)

        tk.Label(frame_superior, text="Selección de Imágenes para Análisis",
                 font=("Arial", 14, "bold"), bg='white').pack(pady=10)

        # Opción: Simulaciones vs Archivos
        self.modo_imagenes = tk.StringVar(value="simulaciones")
        frame_modo = tk.Frame(frame_superior, bg='white')
        frame_modo.pack(fill='x', padx=20, pady=10)

        tk.Radiobutton(frame_modo, text="Usar Simulaciones", variable=self.modo_imagenes,
                       value="simulaciones", command=self.cambiar_modo_imagenes,
bg='white').pack(side='left', padx=20)
                tk.Radiobutton(frame_modo, text="Cargar Imágenes de Archivo",
variable=self.modo_imagenes,
                       value="archivos", command=self.cambiar_modo_imagenes,
bg='white').pack(side='left', padx=20)

        # Frame para opciones específicas
        self.frame_opciones_img = tk.Frame(frame_superior, bg='white')
        self.frame_opciones_img.pack(fill='x', padx=20, pady=10)

        # Inicializar con modo simulaciones
        self.crear_opciones_simulaciones()

        # Botón para realizar análisis
        tk.Button(frame_superior, text="Realizar Análisis KS",
                  command=self.realizar_analisis_ks, bg='#E91E63', fg='white',
                  font=("Arial", 12, "bold")).pack(pady=15)

        # Frame para mostrar imágenes y resultados
        frame_resultados = tk.Frame(frame_ks, bg='white', relief='sunken', bd=2)
        frame_resultados.pack(fill='both', expand=True, padx=10, pady=10)

        # Dividir en dos columnas: imágenes y resultados
        self.frame_imagenes_ks = tk.Frame(frame_resultados, bg='white')
        self.frame_imagenes_ks.pack(side='left', fill='both', expand=True)
```

```
        self.frame_resultados_ks = tk.Frame(frame_resultados, bg='#f8f8f8', width=300)
        self.frame_resultados_ks.pack(side='right', fill='y', padx=10, pady=10)

        # Labels para imágenes
        tk.Label(self.frame_imagenes_ks, text="Imágenes a Comparar",
                 font=("Arial", 12, "bold"), bg='white').pack(pady=10)

        self.label_resultado_ks = tk.Label(self.frame_resultados_ks,
                                           text="Resultados del análisis aparecerán aquí",
                                           bg='#f8f8f8', justify='left', wraplength=280)
        self.label_resultado_ks.pack(anchor='nw', padx=10, pady=10)

    def cambiar_modo_imagenes(self):
        """Cambia entre modo simulaciones y archivos."""
        # Limpiar frame de opciones
        for widget in self.frame_opciones_img.winfo_children():
            widget.destroy()

        if self.modo_imagenes.get() == "simulaciones":
            self.crear_opciones_simulaciones()
        else:
            self.crear_opciones_archivos()

    def crear_opciones_simulaciones(self):
        """Crea las opciones para usar simulaciones."""
        tk.Label(self.frame_opciones_img, text="Selecciona los tipos de tejido a comparar:",
                 bg='white').pack(anchor='w', pady=5)

        frame_tipos = tk.Frame(self.frame_opciones_img, bg='white')
        frame_tipos.pack(fill='x', pady=5)

        # Primera imagen
        frame_img1 = tk.Frame(frame_tipos, bg='white')
        frame_img1.pack(side='left', fill='x', expand=True)
                    tk.Label(frame_img1,    text="Imagen    1:",    bg='white',    font=("Arial",    10,
"bold")).pack(anchor='w')
        self.tipo1_var = tk.StringVar(value="normal")
             tk.Radiobutton(frame_img1,   text="Normal",   variable=self.tipo1_var,   value="normal",
bg='white').pack(anchor='w')
       tk.Radiobutton(frame_img1, text="Osteopenia", variable=self.tipo1_var, value="osteopenia",
bg='white').pack(anchor='w')
```

```
                tk.Radiobutton(frame_img1, text="Osteoporosis", variable=self.tipo1_var,
value="osteoporosis", bg='white').pack(anchor='w')

    # Segunda imagen
    frame_img2 = tk.Frame(frame_tipos, bg='white')
    frame_img2.pack(side='right', fill='x', expand=True)
                tk.Label(frame_img2, text="Imagen 2:", bg='white', font=("Arial", 10,
"bold")).pack(anchor='w')
    self.tipo2_var = tk.StringVar(value="osteoporosis")
        tk.Radiobutton(frame_img2, text="Normal", variable=self.tipo2_var, value="normal",
bg='white').pack(anchor='w')
    tk.Radiobutton(frame_img2, text="Osteopenia", variable=self.tipo2_var, value="osteopenia",
bg='white').pack(anchor='w')
                tk.Radiobutton(frame_img2, text="Osteoporosis", variable=self.tipo2_var,
value="osteoporosis", bg='white').pack(anchor='w')

  def crear_opciones_archivos(self):
    """Crea las opciones para cargar archivos."""
    # Botones para cargar imágenes
    frame_carga = tk.Frame(self.frame_opciones_img, bg='white')
    frame_carga.pack(fill='x', pady=10)

    tk.Button(frame_carga, text="Cargar Imagen 1",
          command=lambda: self.cargar_imagen(1), bg='#4CAF50', fg='white').pack(side='left',
padx=10)
    tk.Button(frame_carga, text="Cargar Imagen 2",
          command=lambda: self.cargar_imagen(2), bg='#4CAF50', fg='white').pack(side='left',
padx=10)

    # Labels para mostrar archivos cargados
    self.label_archivo1 = tk.Label(self.frame_opciones_img, text="Imagen 1: No cargada",
                bg='white', fg='#666')
    self.label_archivo1.pack(anchor='w', padx=10, pady=2)

    self.label_archivo2 = tk.Label(self.frame_opciones_img, text="Imagen 2: No cargada",
                bg='white', fg='#666')
    self.label_archivo2.pack(anchor='w', padx=10, pady=2)

  def cargar_imagen(self, numero):
    """Carga una imagen desde archivo."""
    archivo = filedialog.askopenfilename(
      title=f"Seleccionar Imagen {numero}",
```

```
        filetypes=[("Imágenes", "*.jpg *.jpeg *.png *.bmp *.tiff"), ("Todos", "*.*")]
    )

    if archivo:
        try:
            imagen = cv2.imread(archivo, cv2.IMREAD_GRAYSCALE)
            if imagen is None:
                raise ValueError("No se pudo cargar la imagen")

            if numero == 1:
                self.imagen1 = imagen
                self.label_archivo1.config(text=f"Imagen 1: {archivo.split('/')[-1]}", fg='#4CAF50')
            else:
                self.imagen2 = imagen
                self.label_archivo2.config(text=f"Imagen 2: {archivo.split('/')[-1]}", fg='#4CAF50')

        except Exception as e:
            messagebox.showerror("Error", f"Error al cargar imagen: {str(e)}")

def generar_imagen_individual(self):
    """Genera una imagen individual según parámetros."""
    try:
        ancho = int(self.ancho_var.get())
        alto = int(self.alto_var.get())
        semilla = int(self.semilla_var.get()) if self.semilla_var.get() else None

        # Convertir string a TipoTejido
        tipo = gt.TipoTejido(self.tipo_tejido.get())

        # Generar imagen usando el generador de tejidos
        imagen = gt.generar_imagen_hueso_simulada(
            width=ancho,
            height=alto,
            tipo=tipo,
            incluir_cortical=self.incluir_cortical.get(),
            seed=semilla
        )

        self.mostrar_imagen_simulada(imagen, f"Tejido {tipo.value.capitalize()}")

    except ValueError as e:
        messagebox.showerror("Error", f"Error en los parámetros: {str(e)}")
```

```
    except Exception as e:
        messagebox.showerror("Error", f"Error al generar imagen: {str(e)}")

def generar_dataset(self):
    """Genera un dataset completo."""
    try:
        # Obtener parámetros
        ancho = int(self.ancho_var.get())
        alto = int(self.alto_var.get())
        semilla = int(self.semilla_var.get()) if self.semilla_var.get() else None

        # Crear directorio para el dataset si no existe
        import os
        directorio = filedialog.askdirectory(title="Seleccionar directorio para dataset")
        if not directorio:
            return

        # Generar dataset
        dataset = gt.generar_dataset_simulado(
            n_imagenes_por_tipo=5,  # Número de imágenes por tipo
            dimensiones=(ancho, alto),
            incluir_cortical=self.incluir_cortical.get(),
            seed_base=semilla
        )

        # Guardar imágenes
        for tipo, imagenes in dataset.items():
            tipo_dir = os.path.join(directorio, tipo)
            os.makedirs(tipo_dir, exist_ok=True)

            for i, imagen in enumerate(imagenes):
                ruta = os.path.join(tipo_dir, f"{tipo}_{i+1}.png")
                cv2.imwrite(ruta, imagen)

        messagebox.showinfo("Éxito", f"Dataset generado en:\n{directorio}")

    except Exception as e:
        messagebox.showerror("Error", f"Error al generar dataset: {str(e)}")

def comparar_tipos(self):
    """Genera y muestra comparación de los tres tipos."""
    try:
```

```
# Limpiar el frame
for widget in self.frame_visualizacion.winfo_children():
  widget.destroy()

tk.Label(self.frame_visualizacion, text="Comparación de Tipos de Tejido",
      font=("Arial", 14, "bold"), bg='white').pack(pady=10)

frame_imagenes = tk.Frame(self.frame_visualizacion, bg='white')
frame_imagenes.pack()

ancho = int(self.ancho_var.get())
alto = int(self.alto_var.get())
semilla = int(self.semilla_var.get()) if self.semilla_var.get() else None

for i, tipo in enumerate(gt.TipoTejido):
  frame_tipo = tk.Frame(frame_imagenes, bg='white')
  frame_tipo.pack(side='left', padx=10)

  # Generar imagen usando el generador
  imagen = gt.generar_imagen_hueso_simulada(
    width=ancho,
    height=alto,
    tipo=tipo,
    incluir_cortical=self.incluir_cortical.get(),
    seed=semilla + i
  )

  imagen_pil = Image.fromarray(imagen)
  imagen_pil = imagen_pil.resize((200, 200), Image.Resampling.LANCZOS)
  imagen_tk = ImageTk.PhotoImage(imagen_pil)

  tk.Label(frame_tipo, text=tipo.value.capitalize(),
       font=("Arial", 10, "bold"), bg='white').pack()
  label_img = tk.Label(frame_tipo, image=imagen_tk, bg='white')
  label_img.image = imagen_tk
  label_img.pack()

  # Mostrar estadísticas básicas
  stats_text = f"Media: {imagen.mean():.1f}\nStd: {imagen.std():.1f}"
  tk.Label(frame_tipo, text=stats_text,
       font=("Arial", 8), bg='white').pack()
```

```
    except Exception as e:
        messagebox.showerror("Error", f"Error al comparar tipos: {str(e)}")

def realizar_analisis_ks(self):
    """Realiza el análisis KS según el modo seleccionado."""
    try:
        if self.modo_imagenes.get() == "simulaciones":
            self.generar_imagenes_para_ks()

        if self.imagen1 is None or self.imagen2 is None:
            messagebox.showerror("Error", "Faltan imágenes para el análisis")
            return

        # Realizar prueba KS
        self.resultados_ks = ks.prueba_ks_imagenes(self.imagen1, self.imagen2)

        # Mostrar resultados
        self.mostrar_resultados_ks(self.resultados_ks)
        self.mostrar_imagenes_ks()

    except Exception as e:
        messagebox.showerror("Error", f"Error en análisis KS: {str(e)}")

def generar_imagenes_para_ks(self):
    """Genera las imágenes simuladas para el análisis KS."""
    try:

        ancho = int(self.ancho_var.get())
        alto = int(self.alto_var.get())

        # Convertir strings a TipoTejido
        tipo1 = gt.TipoTejido(self.tipo1_var.get())
        tipo2 = gt.TipoTejido(self.tipo2_var.get())

        # Generar imágenes
        semilla = int(self.semilla_var.get()) if self.semilla_var.get() else None

        self.imagen1 = gt.generar_imagen_hueso_simulada(
            width=ancho, height=alto,
            tipo=tipo1,
            incluir_cortical=self.incluir_cortical.get(),
            seed=semilla
```

```
        )

        self.imagen2 = gt.generar_imagen_hueso_simulada(
            width=ancho, height=alto,
            tipo=tipo2,
            incluir_cortical=self.incluir_cortical.get(),
            seed=semilla + 1  # Asegurar que sea diferente
        )

    except Exception as e:
        messagebox.showerror("Error", f"Error al generar imágenes: {str(e)}")
        raise e

def mostrar_imagen_simulada(self, imagen, titulo):
    """Muestra una imagen simulada en el área de visualización."""
    # Limpiar el frame
    for widget in self.frame_visualizacion.winfo_children():
        widget.destroy()

    # Convertir imagen para Tkinter
    imagen_pil = Image.fromarray(imagen)
    imagen_pil = imagen_pil.resize((300, 300), Image.Resampling.LANCZOS)
    imagen_tk = ImageTk.PhotoImage(imagen_pil)

    # Mostrar imagen
    tk.Label(self.frame_visualizacion, text=titulo, font=("Arial", 12, "bold"),
             bg='white').pack(pady=10)
    label_img = tk.Label(self.frame_visualizacion, image=imagen_tk, bg='white')
    label_img.image = imagen_tk  # Mantener referencia
    label_img.pack()

def mostrar_comparacion_tipos(self):
    """Muestra comparación de los tres tipos de tejido."""
    # Limpiar el frame
    for widget in self.frame_visualizacion.winfo_children():
        widget.destroy()

    tk.Label(self.frame_visualizacion, text="Comparación de Tipos de Tejido",
             font=("Arial", 14, "bold"), bg='white').pack(pady=10)

    frame_imagenes = tk.Frame(self.frame_visualizacion, bg='white')
    frame_imagenes.pack()
```

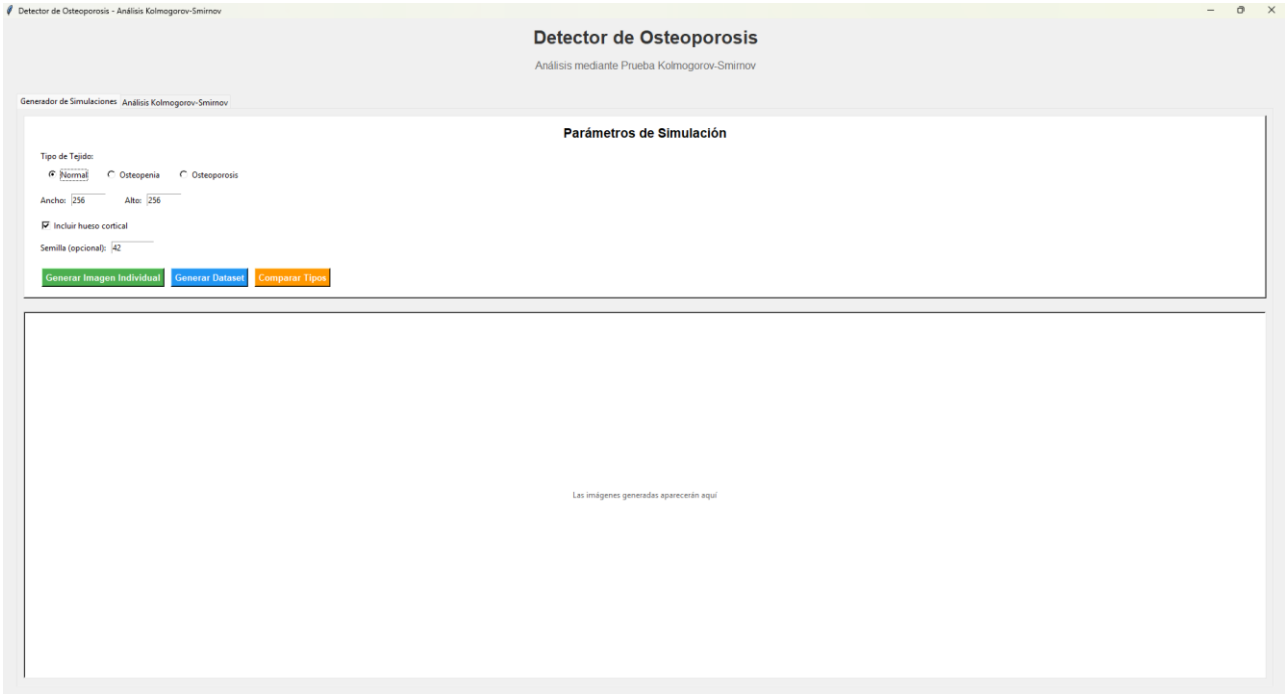
```
    tipos = ["Normal", "Osteopenia", "Osteoporosis"]
    for i, tipo in enumerate(tipos):
        frame_tipo = tk.Frame(frame_imagenes, bg='white')
        frame_tipo.pack(side='left', padx=10)

        # Simular imagen
        np.random.seed(40 + i)
        imagen = np.random.randint(50 + i*30, 200 - i*30, (128, 128), dtype=np.uint8)

        imagen_pil = Image.fromarray(imagen)
        imagen_pil = imagen_pil.resize((150, 150), Image.Resampling.LANCZOS)
        imagen_tk = ImageTk.PhotoImage(imagen_pil)

        tk.Label(frame_tipo, text=tipo, font=("Arial", 10, "bold"), bg='white').pack()
        label_img = tk.Label(frame_tipo, image=imagen_tk, bg='white')
        label_img.image = imagen_tk
        label_img.pack()

def mostrar_imagenes_ks(self):
    """Muestra las imágenes que se están comparando y sus gráficos."""
    # Limpiar frame de imágenes
    for widget in self.frame_imagenes_ks.winfo_children():
        widget.destroy()

    # Frame para imágenes
    frame_imgs = tk.Frame(self.frame_imagenes_ks, bg='white')
    frame_imgs.pack(fill='x', pady=10)

    # Mostrar ambas imágenes
    for i, (imagen, titulo) in enumerate([(self.imagen1, "Imagen 1"), (self.imagen2, "Imagen 2")]):
        frame_img = tk.Frame(frame_imgs, bg='white')
        frame_img.pack(side='left', padx=20)

        imagen_pil = Image.fromarray(imagen)
        imagen_pil = imagen_pil.resize((200, 200), Image.Resampling.LANCZOS)
        imagen_tk = ImageTk.PhotoImage(imagen_pil)

        tk.Label(frame_img, text=titulo, font=("Arial", 10, "bold"), bg='white').pack()
        label_img = tk.Label(frame_img, image=imagen_tk, bg='white')
        label_img.image = imagen_tk
        label_img.pack()
```

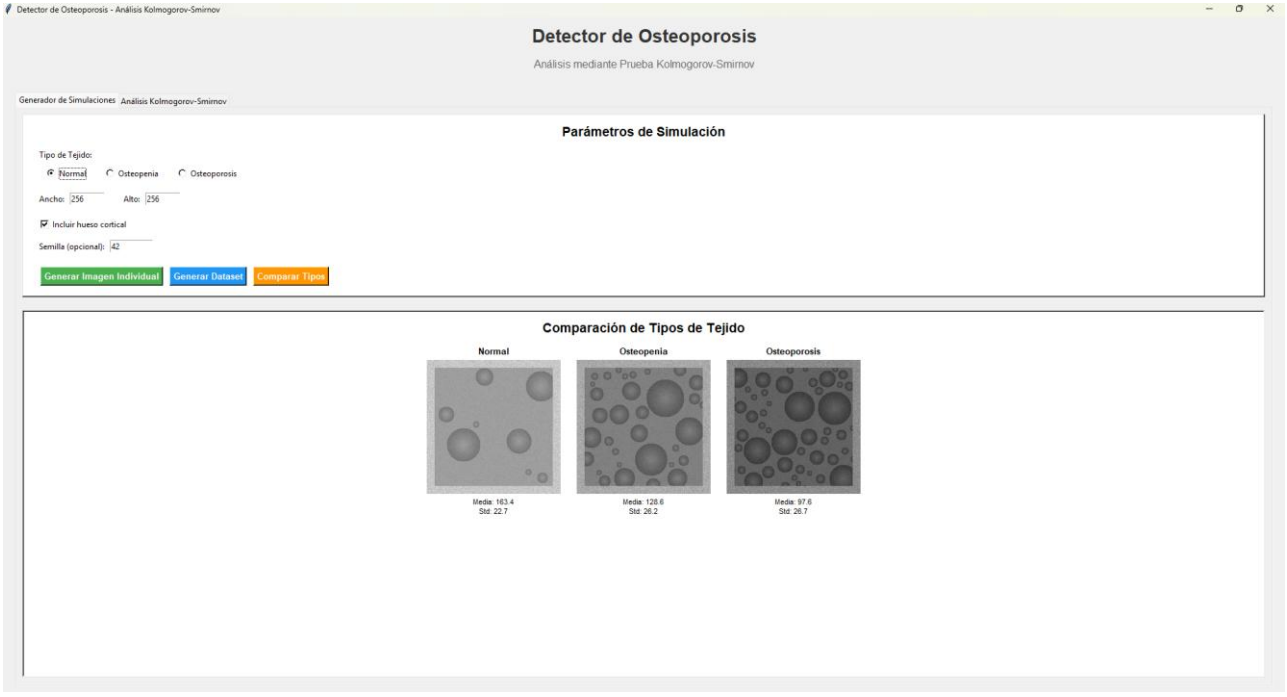
```
# Frame para gráficos
frame_graficos = tk.Frame(self.frame_imagenes_ks, bg='white')
frame_graficos.pack(fill='both', expand=True, pady=10)

# Crear figura con dos subplots
fig = Figure(figsize=(10, 4))

# Histogramas
ax1 = fig.add_subplot(121)
hist1 = ks.calcular_histograma_normalizado(self.imagen1)
hist2 = ks.calcular_histograma_normalizado(self.imagen2)
x = np.arange(256)

ax1.plot(x, hist1, label='Imagen 1', color='blue')
ax1.plot(x, hist2, label='Imagen 2', color='red', linestyle='--')
ax1.set_title('Histogramas')
ax1.set_xlabel('Nivel de gris')
ax1.set_ylabel('Frecuencia relativa')
ax1.legend()

# FDA
ax2 = fig.add_subplot(122)
fda1 = ks.calcular_fda_empirica(hist1)
fda2 = ks.calcular_fda_empirica(hist2)

ax2.plot(x, fda1, label='Imagen 1', color='blue')
ax2.plot(x, fda2, label='Imagen 2', color='red', linestyle='--')
ax2.set_title('Función de Distribución Acumulada')
ax2.set_xlabel('Nivel de gris')
ax2.set_ylabel('Probabilidad acumulada')
ax2.legend()

# Mostrar D máximo
if hasattr(self, 'resultados_ks'):
   d_max = self.resultados_ks['estadistico_D']
   idx_max = np.argmax(np.abs(fda1 - fda2))
   ax2.plot([x[idx_max], x[idx_max]],
         [min(fda1[idx_max], fda2[idx_max]),
          max(fda1[idx_max], fda2[idx_max])],
         'k--', label=f'D={d_max:.3f}')
   ax2.legend()
```

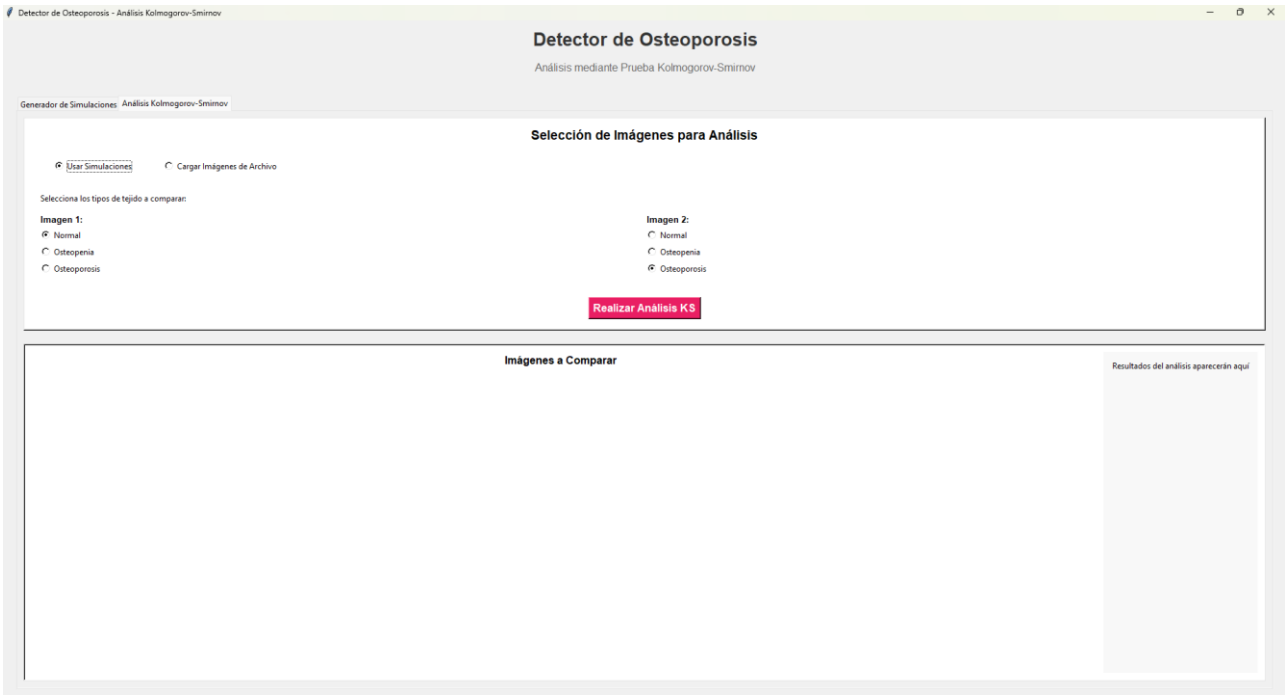
```
        fig.tight_layout()

        # Crear canvas y mostrarlo
        canvas = FigureCanvasTkAgg(fig, master=frame_graficos)
        canvas.draw()
        canvas.get_tk_widget().pack(fill='both', expand=True)

    def mostrar_resultados_ks(self, resultados):
        """Muestra los resultados del análisis KS."""
        # Limpiar resultados anteriores
        for widget in self.frame_resultados_ks.winfo_children():
            widget.destroy()

        # Título
        tk.Label(self.frame_resultados_ks, text="Resultados del Test KS",
                 font=("Arial", 12, "bold"), bg='#f8f8f8').pack(pady=10)

        # Estadísticas principales
        stats_text = f"""
Estadístico D:  {resultados['estadistico_D']:.4f}
Lambda KS:     {resultados['lambda_KS']:.4f}
P-valor:       {resultados['p_valor']:.4f}
Alpha:         {resultados['alpha']:.4f}

Decisión:
{"Se rechaza H₀" if resultados['rechazar_H0'] else "No se rechaza H₀"}
"""

        tk.Label(self.frame_resultados_ks, text=stats_text,
                 font=("Courier", 10), bg='#f8f8f8', justify='left').pack(padx=10)

        # Conclusión
        conclusion_frame = tk.Frame(self.frame_resultados_ks, bg='#f8f8f8')
        conclusion_frame.pack(fill='x', padx=10, pady=10)

        color = '#d32f2f' if resultados['rechazar_H0'] else '#388e3c'
        tk.Label(conclusion_frame, text="Conclusión:",
                 font=("Arial", 10, "bold"), bg='#f8f8f8').pack(anchor='w')
        tk.Label(conclusion_frame, text=resultados['conclusion'],
                 font=("Arial", 9), bg='#f8f8f8', fg=color,
                 wraplength=250, justify='left').pack(pady=(5,0))
```

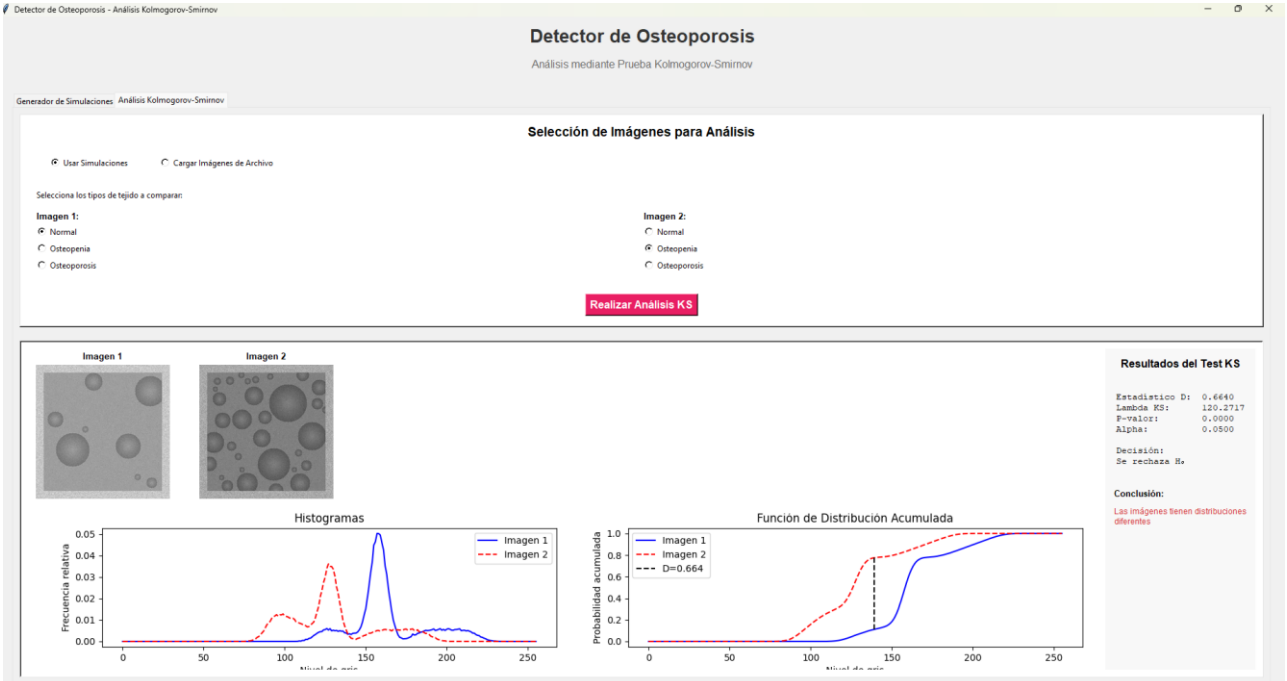
```
def main():
    """Función principal para ejecutar la aplicación."""
    root = tk.Tk()
    app = DetectorOsteoporosisApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

UI


Detector de Osteoporosis - Análisis Kolmogorov-Smirnov
Detector de Osteoporosis
Análisis mediante Prueba Kolmogorov-Smirnov
Generador de Simulaciones
Análisis Kolmogorov-Smirnov
Parámetros de Simulación
Tipo de Tejido:
Normal
Osteopenia
Osteoporosis
Ancho: 256
Alto: 256
Incluir hueso cortical
Semilla (opcional): 42
Generar Imagen Individual
Generar Dataset
Comparar Tipos
Las imágenes generadas aparecerán aquí


Detector de Osteoporosis - Análisis Kolmogorov-Smirnov
Detector de Osteoporosis
Análisis mediante Prueba Kolmogorov-Smirnov
Generador de Simulaciones
Análisis Kolmogorov-Smirnov
Parámetros de Simulación
Tipo de Tejido:
Normal
Osteopenia
Osteoporosis
Ancho: 256
Alto: 256
Incluir hueso cortical
Semilla (opcional): 42
Generar Imagen Individual
Generar Dataset
Comparar Tipos
Comparación de Tipos de Tejido
Normal
Osteopenia
Osteoporosis
Media: 163.4
Std: 22.7
Media: 128.6
Std: 26.2
Media: 97.6
Std: 26.7


Detector de Osteoporosis - Análisis Kolmogorov-Smirnov
Detector de Osteoporosis
Análisis mediante Prueba Kolmogorov-Smirnov
Generador de Simulaciones
Análisis Kolmogorov-Smirnov
Selección de Imágenes para Análisis
Usar Simulaciones
Cargar Imágenes de Archivo
Selecciona los tipos de tejido a comparar:
Imagen 1:
Normal
Osteopenia
Osteoporosis
Imagen 2:
Normal
Osteopenia
Osteoporosis
Realizar Análisis KS
Imágenes a Comparar
Resultados del análisis aparecerán aquí


Detector de Osteoporosis - Análisis Kolmogorov-Smirnov
Detector de Osteoporosis
Análisis mediante Prueba Kolmogorov-Smirnov
Generador de Simulaciones
Análisis Kolmogorov-Smirnov
Selección de Imágenes para Análisis
Usar Simulaciones
Cargar Imágenes de Archivo
Selecciona los tipos de tejido a comparar:
Imagen 1:
Normal
Osteopenia
Osteoporosis
Imagen 2:
Normal
Osteopenia
Osteoporosis
Realizar Análisis KS
Imagen 1
Imagen 2
Histogramas
Frecuencia relativa
Imagen 1
Imagen 2
Función de Distribución Acumulada
Probabilidad acumulada
Imagen 1
Imagen 2
D=0.664
Resultados del Test KS
Estadístico D: 0.6640
Lambda KS: 120.2717
P-valor: 0.0000
Alpha: 0.0500
Decisión:
Se rechaza H₀
Conclusión:
Las imágenes tienen distribuciones diferentes

Glosario de términos

Término	Definición
KS (Kolmogorov-Smirnov)	Prueba estadística que compara la distribución de dos muestras.
valor-p	Probabilidad de observar un resultado al menos tan extremo como el obtenido, si la hipótesis nula fuera cierta.
λ_KS	Estadístico intermedio utilizado para calcular el valor-p en la prueba KS.
Simulación	Generación artificial de imágenes para entrenamiento/pruebas.
Dataset	Conjunto de datos estructurado, en este caso, imágenes por clase.
GUI	Interfaz Gráfica de Usuario (Tkinter en este proyecto).
Imagen corrupta	Archivo de imagen ilegible o con formato inválido.
Tkinter	Biblioteca de Python para construir interfaces gráficas.

Referencias bibliográficas

Demidenko, E. (2004). Kolmogorov-Smirnov Test for Image Comparison. En A. Laganá, M. L. Gavrilova, V. Kumar, Y. Mun, C. J. K. Tan, & O. Gervasi (Eds.), *Computational Science and Its Applications – ICCSA 2004* (Vol. 3046, pp. 1023–1031). Springer. https://doi.org/10.1007/978-3-540-24768-5_100

Jang, R., Choi, J. H., Kim, N., Chang, J. S., Yoon, P. W., & Kim, C.-H. (2021). Prediction of osteoporosis from simple hip radiography using deep learning algorithm. *Scientific Reports, 11*, Article 19997. https://doi.org/10.1038/s41598-021-99549-6

Jalammar, J. (s.f.). *Visual numpy*. Retrieved May 26, 2025, from https://jalammar.github.io/visual-numpy/