Arquitectura de archivos

Cada script tiene responsabilidades claras y específicas, lo cual facilita la mantenibilidad y escalabilidad del proyecto. A continuación, un resumen y recomendaciones para cada uno de los componentes:

# 1. detector\_neumonia.py

* **Funcionalidad**: Contiene el diseño de la interfaz gráfica utilizando Tkinter. Los botones llaman métodos contenidos en otros scripts.
* **Recomendación**: Asegúrate de que la interfaz gráfica sea intuitiva y fácil de usar. Considera la posibilidad de agregar mensajes de error y confirmación para mejorar la experiencia del usuario.

## Explicación del Código

1. **Importaciones**: Se importan los módulos necesarios, incluyendo Tkinter, PIL para el manejo de imágenes y pydicom para leer imágenes DICOM. También se importa el módulo integrator.
2. **Clase PneumoniaDetectorApp**:
   * Inicializa la ventana principal de Tkinter y configura los elementos de la interfaz (botones, etiquetas, etc.).
   * upload\_image: Función que permite al usuario cargar una imagen DICOM desde su sistema de archivos y mostrarla en la interfaz.
   * predict: Función que llama al módulo integrator para obtener la predicción y el mapa de calor, y muestra los resultados en la interfaz.
3. **Función Principal**: Crea una instancia de tk.Tk() y de la clase PneumoniaDetectorApp, y ejecuta el bucle principal de Tkinter.

## Notas

* Asegúrate de que el módulo integrator y los demás scripts estén en el mismo directorio o en el PYTHONPATH.
* integrator.process\_image(self.image\_path) es una llamada hipotética que deberás ajustar según la implementación de tu módulo integrator.

Este código proporciona una base sólida para la interfaz gráfica de tu proyecto. Puedes ajustarlo y expandirlo según las necesidades específicas de tu aplicación.

# 2. integrator.py

* **Funcionalidad**: Integra los demás scripts y retorna solamente lo necesario para ser visualizado en la interfaz gráfica. Retorna la clase, la probabilidad y un mapa de calor generado por Grad-CAM.
* **Recomendación**: Mantén este módulo lo más limpio posible, actuando solo como intermediario entre la interfaz gráfica y la lógica de negocio. Implementa un buen manejo de excepciones para capturar y manejar posibles errores.

## Explicación del Código

1. **Importaciones**: Se importan los módulos read\_img, preprocess\_img, load\_model, y grad\_cam.
2. **Función process\_image**:
   * **Leer la imagen DICOM**: Se utiliza la función read\_dicom del módulo read\_img para leer la imagen en formato DICOM.
   * **Preprocesar la imagen**: Se utiliza la función preprocess del módulo preprocess\_img para preprocesar la imagen.
   * **Cargar el modelo**: Se utiliza la función load\_cnn\_model del módulo load\_model para cargar el modelo de red neuronal convolucional previamente entrenado.
   * **Obtener la predicción y el mapa de calor**: Se utiliza la función generate\_grad\_cam del módulo grad\_cam para obtener la predicción y el mapa de calor generado por Grad-CAM.
   * **Guardar el mapa de calor**: Se guarda el mapa de calor en un archivo PNG.
3. **Bloque Principal**:
   * **Prueba del script**: Se incluye una sección para probar el script con una imagen de ejemplo. Esta sección se puede eliminar o comentar cuando el script se utilice como parte de la aplicación principal.

## Notas

* **Funciones en otros módulos**: Asegúrate de que las funciones read\_dicom, preprocess, load\_cnn\_model, y generate\_grad\_cam estén correctamente implementadas en sus respectivos módulos.
* **Rutas de archivos**: Ajusta las rutas de los archivos según sea necesario. En este ejemplo, se asume que el modelo WilhemNet86.h5 y la imagen example.dcm están en el mismo directorio que integrator.py.

# 3. read\_img.py

* **Funcionalidad**: Lee la imagen en formato DICOM y la convierte a un arreglo para su preprocesamiento.
* **Recomendación**: Asegúrate de que este script maneje correctamente diferentes variantes de archivos DICOM y que las imágenes se leen de manera eficiente.

### Explicación del Código

1. **Importaciones**: Se importan los módulos necesarios, incluyendo pydicom para leer imágenes DICOM, numpy para manipular arreglos y matplotlib.pyplot para mostrar imágenes.
2. **Función read\_dicom**:
   * **Parámetros**: Toma la ruta del archivo DICOM como entrada.
   * **Proceso**: Lee el archivo DICOM usando pydicom.dcmread() y convierte la imagen en un arreglo numpy.
   * **Manejo de errores**: Si ocurre un error al leer el archivo DICOM, se captura y se imprime un mensaje de error.
   * **Retorno**: Devuelve el arreglo numpy de la imagen o None si ocurre un error.
3. **Función show\_image**:
   * **Parámetros**: Toma el arreglo numpy de la imagen como entrada.
   * **Proceso**: Muestra la imagen usando matplotlib.pyplot.imshow().
4. **Bloque Principal**:
   * **Prueba del script**: Se incluye una sección para probar el script con una imagen de ejemplo. Lee la imagen DICOM y la muestra. Esta sección se puede eliminar o comentar cuando el script se utilice como parte de la aplicación principal.

### Notas

* **Dependencias**: Asegúrate de tener instalados los módulos pydicom, numpy, y matplotlib.
* **Manejo de errores**: El manejo de errores puede ajustarse según las necesidades de tu aplicación. Aquí, simplemente se imprime un mensaje de error.

# 4. preprocess\_img.py

* **Funcionalidad**: Procesa la imagen: resize a 512x512, conversión a escala de grises, ecualización del histograma con CLAHE, normalización de la imagen entre 0 y 1, y conversión del arreglo de imagen a formato de batch (tensor).
* **Recomendación**: Verifica que cada paso del preprocesamiento sea necesario y esté correctamente implementado. Mantén las funciones modulares para facilitar los ajustes futuros.

### Explicación del Código

1. **Importaciones**: Se importan los módulos necesarios, incluyendo cv2 para el procesamiento de imágenes, numpy para la manipulación de arreglos y torch para trabajar con tensores.
2. **Función preprocess**:
   * **Parámetros**: Toma el arreglo numpy de la imagen como entrada.
   * **Resize**: Redimensiona la imagen a 512x512 píxeles.
   * **Conversión a escala de grises**: Convierte la imagen a escala de grises si no lo está ya.
   * **Ecualización del histograma con CLAHE**: Aplica la ecualización del histograma para mejorar el contraste.
   * **Normalización**: Normaliza la imagen para que sus valores estén entre 0 y 1.
   * **Conversión a tensor**: Convierte la imagen normalizada en un tensor de PyTorch y añade dos dimensiones para el batch y el canal.
   * **Retorno**: Devuelve el tensor de la imagen preprocesada.
3. **Bloque Principal**:
   * **Prueba del script**: Incluye una sección para probar el script con una imagen de ejemplo. Lee la imagen DICOM usando el script read\_img.py, la preprocesa y muestra la forma del tensor resultante.

### Notas

* **Dependencias**: Asegúrate de tener instalados los módulos cv2, numpy, y torch.
* **Manejo de errores**: El manejo de errores puede ajustarse según las necesidades de tu aplicación. Aquí, simplemente se imprime un mensaje de error.

# 5. load\_model.py

* **Funcionalidad**: Lee el archivo binario del modelo de red neuronal convolucional previamente entrenado llamado 'WilhemNet86.h5'.
* **Recomendación**: Incluye opciones para cargar diferentes versiones del modelo, si es necesario. Implementa la carga del modelo de forma segura para evitar errores durante la predicción.

### Explicación del Código

1. **Importaciones**: Se importan los módulos necesarios, incluyendo torch y torch.nn.
2. **Clase WilhemNet86**:
   * **Inicialización**: Define la arquitectura del modelo de red neuronal convolucional. Las capas deben coincidir con la arquitectura del modelo que se ha entrenado previamente.
   * **Forward**: Define el paso hacia adelante a través de la red.
3. **Función load\_cnn\_model**:
   * **Parámetros**: Toma la ruta del archivo del modelo como entrada.
   * **Inicialización del modelo**: Crea una instancia de la clase WilhemNet86.
   * **Carga del estado del modelo**: Carga los pesos del modelo desde el archivo especificado usando torch.load() y asigna estos pesos al modelo inicializado.
   * **Modo de evaluación**: Pone el modelo en modo evaluación con model.eval().
   * **Retorno**: Devuelve el modelo cargado.
4. **Bloque Principal**:
   * **Prueba del script**: Incluye una sección para probar el script con un modelo de ejemplo. Carga el modelo desde la ruta especificada y verifica si se ha cargado correctamente.

### Notas

* **Dependencias**: Asegúrate de tener instalado PyTorch.
* **Arquitectura del modelo**: La definición de la clase WilhemNet86 debe coincidir con la arquitectura del modelo que fue entrenado. Ajusta las capas y parámetros según sea necesario.
* **Ruta del modelo**: Asegúrate de que la ruta al archivo del modelo es correcta y que el archivo está en el formato esperado (por ejemplo, .pth o .pt).

# 6. grad\_cam.py

* **Funcionalidad**: Recibe la imagen y la procesa, carga el modelo, obtiene la predicción y la capa convolucional de interés para obtener las características relevantes de la imagen.
* **Recomendación**: Asegúrate de que el Grad-CAM esté bien implementado y que los mapas de calor generados sean precisos y útiles para la interpretación. Prueba este script extensivamente con diferentes imágenes.

Esta estructura modular te permitirá desarrollar y probar cada componente de manera independiente antes de integrarlos en la aplicación completa.