**Histology Bone Analyzer - Explicación de Códigos**

Este documento explica la función de cada archivo de código relevante en el proyecto Phygital Human Bone, organizado por categorías para facilitar su comprensión.

**Aplicaciones Principales**

**Detección de Canales de Havers**

* **detection\_app.py** - Aplicación principal que detecta canales de Havers en imágenes histológicas usando YOLO. Incluye interfaz gráfica, segmentación de imágenes y visualización de resultados.
  + **Evolución**: detection\_app\_iteration1.py → detection\_app\_iteration2.py → detection\_app\_iteration3.py → detection\_app.py (versión actual)
  + **Función principal**: Identificar y analizar canales de Havers en imágenes histológicas óseas

**Análisis Espacial**

* **breaking\_app.py** - Analiza la distribución espacial de canales de Havers dividiendo la imagen en 9 cuadrantes. Identifica zonas de mayor concentración y calcula estadísticas por región.
  + **Flujo de trabajo**: Toma los resultados de detection\_app.py, reconstruye las detecciones, y realiza un análisis por cuadrantes
  + **Salida**: Imagen con cuadrantes marcados y estadísticas por región

**Distribución Paramétrica**

* **distribution\_app.py** - Genera distribuciones paramétrica de osteonas para diferentes secciones del fémur, con propiedades configurables por sección (epífisis, metáfisis, diáfisis).
  + **Propósito**: Crear datos para Grasshopper que permitan modelar huesos con distribuciones realistas de osteonas
  + **Interfaz**: GUI con pestañas para parámetros, visualización y exportación

**Scripts de Utilidad**

**Procesamiento de Imágenes**

* **topng5mb.py** - Convierte imágenes TIF a PNG manteniendo un tamaño máximo de 3.5MB mediante redimensionamiento inteligente. Crucial para preparar imágenes para análisis.
* **imagetored.py** - Convierte tonos azules a rojos en imágenes (útil para matrices de confusión y visualizaciones).

**Verificación de Entorno**

* **check\_gpu.py** - Comprueba si CUDA está disponible y qué GPU está siendo detectada para optimizar el rendimiento del modelo YOLO.
* **tinker\_check.py** - Verifica que Tkinter funciona correctamente para asegurar que las interfaces gráficas serán compatibles con el sistema.

**Desarrollo y Entrenamiento**

* **check\_projects.py** - Script que verifica y lista los proyectos disponibles en un workspace de Roboflow usando su API.
* **inference\_local.py** - Realiza inferencias locales de detección con el modelo YOLO entrenado, sin necesidad de la aplicación completa.

**Versiones Históricas**

**Desarrollo Inicial**

* **phygital-codeosteonas.py** - Prototipo inicial del programa de detección de canales de Havers.
* **fixed-phygital-code.py** - Versión intermedia con correcciones de errores y mejoras en la visualización.
* **corrected-phygital-code.py** - Versión refinada antes de la reestructuración en módulos separados.

**Organización del Código**

El código del proyecto sigue una estructura modular donde:

1. Las **aplicaciones principales** (detection\_app, breaking\_app, distribution\_app) están en carpetas separadas
2. Cada aplicación tiene versiones históricas documentadas
3. Los **scripts de utilidad** están centralizados en la carpeta scripts
4. La documentación y manuales de usuario están en la carpeta docs

**Notas para Desarrolladores**

* Las aplicaciones usan un modelo YOLO entrenado que debe estar en una de las rutas predefinidas
* El procesamiento de imágenes grandes se realiza mediante segmentación para optimizar la memoria
* La interfaz de usuario utiliza Tkinter para mantener la compatibilidad entre plataformas
* Los resultados intermedios se guardan en carpetas organizadas para facilitar el seguimiento