FUNCIONES PRINCIPALES DE PROCESAMIENTO

1. Función Principal de Ejecución

```
def main_sistema_completo():
    """Función principal del sistema completo - COMPLETAMENTE CORREGIDA"""
    print("🚀 INICIANDO SISTEMA COMPLETO DE ANÁLISIS TIROIDEO")

    # Inicializar componentes
    eda_analyzer = AdvancedEDA()
    bias_analyzer = BiasAnalysis()
    optimizer = HyperparameterOptimizer()

    try:
        # 1. Cargar datos
        X, y, df_metadatos = cargar_dataset_completo_avanzado()

        # 2. Análisis EDA completo
        eda_results = eda_analyzer.perform_comprehensive_eda(X, y, df_metadatos)

        # 3. Análisis de sesgo
        X_features = df_metadatos.select_dtypes(include=[np.number])
        bias_results = bias_analyzer.perform_bias_analysis(X_features, y)

        # 4. Optimización
        optimization_results = optimizer.perform_comprehensive_optimization(
            X_features, y, 'random_forest'
        )

        return {
            'eda_results': eda_results,
            'bias_results': bias_results,
            'optimization_results': optimization_results
        }

    except Exception as e:
        print(f"❌ Error en el sistema completo: {e}")
        return None
```

2. Carga y Preprocesamiento de Imágenes

```
def cargar_dataset_completo_avanzado():
  """Carga completa del dataset con extracción de características"""
  todas_imagenes = []
  todas_etiquetas = []
  todos_metadatos = []

  for clase in CLASSES:
    ruta_clase = os.path.join(BASE_PATH, clase)
    archivos = [f for f in os.listdir(ruta_clase)
          if es_archivo_imagen_avanzado(os.path.join(ruta_clase, f))]

    for archivo in archivos[:MAX_IMAGES_PER_CLASS]:
      ruta_completa = os.path.join(ruta_clase, archivo)
      imagen = cargar_y_preprocesar_imagen_avanzado(ruta_completa)

      if imagen is not None:
        todas_imagenes.append(imagen)
        todas_etiquetas.append(clase)
        caracteristicas = extraer_caracteristicas_avanzadas_completas(imagen)

        metadato = {
          'clase': clase,
          'archivo': archivo,
          'ruta': ruta_completa,
          **caracteristicas
        }
        todos_metadatos.append(metadato)

  X = np.array(todas_imagenes, dtype=np.float32)
  y = np.array(todas_etiquetas)
  df_metadatos = pd.DataFrame(todos_metadatos)

  return X, y, df_metadatos
```

3. Preprocesamiento de Imágenes

```
def cargar_y_preprocesar_imagen_avanzado(ruta, tamaño=IMG_SIZE):
    """Carga y preprocesa una imagen individual"""
    try:
        with Image.open(ruta) as img:
            if img.mode != 'RGB':
                img = img.convert('RGB')
            img = mejorar_calidad_imagen(img)
            img = img.filter(ImageFilter.MedianFilter(size=3))
            img.thumbnail((tamaño[0] * 2, tamaño[1] * 2), Image.Resampling.LANCZOS)
            img = img.resize(tamaño, Image.Resampling.LANCZOS)
            arr = np.array(img, dtype=np.float32) / 255.0
            return arr
    except Exception as e:
        print(f"❌ Error procesando {ruta}: {e}")
        return None

def mejorar_calidad_imagen(imagen):
    """Mejora la calidad de la imagen con ajustes"""
    enhancer = ImageEnhance.Brightness(imagen)
    imagen = enhancer.enhance(0.9)
    enhancer = ImageEnhance.Contrast(imagen)
    imagen = enhancer.enhance(1.1)
    enhancer = ImageEnhance.Sharpness(imagen)
    imagen = enhancer.enhance(1.05)
    return imagen
```

4. Extracción de Características

```
def extraer_caracteristicas_avanzadas_completas(arr):
  """Extrae características avanzadas de la imagen"""
  try:
    import cv2
    gris = np.mean(arr, axis=2)

    # Características básicas
    intensidad = np.mean(gris)
    contraste = np.std(gris)
    entropia = stats.entropy(gris.flatten() + 1e-8)

    # Momentos de Hu
    momentos = cv2.moments((gris * 255).astype(np.uint8))
    hu_momentos = cv2.HuMoments(momentos).flatten()

    # Características de bordes
    bordes_canny = cv2.Canny((gris * 255).astype(np.uint8), 30, 150)
    densidad_bordes = np.mean(bordes_canny > 0)

    # Gradientes
    grad_x = sobel(gris, axis=0)
    grad_y = sobel(gris, axis=1)
    magnitud_gradiente = np.sqrt(grad_x**2 + grad_y**2)

    # Estadísticas
    asimetria = skew(gris.flatten())
    curtosis = kurtosis(gris.flatten())

    return {
      'intensidad_promedio': float(intensidad),
      'contraste': float(contraste),
      'entropia': float(entropia),
      'asimetria': float(asimetria),
      'curtosis': float(curtosis),
      'densidad_bordes': float(densidad_bordes),
      'magnitud_gradiente_promedio': float(np.mean(magnitud_gradiente)),
      'hu_momento_1': float(hu_momentos[0]),
      'hu_momento_2': float(hu_momentos[1]),
      'heterogeneidad': float(contraste / (intensidad + 1e-8))
    }
```

```
    except Exception as e:
        print(f"⚠️ Error en análisis avanzado: {e}")
        return None
```

5. Modelado y Predicción

```
python
def crear_modelo_prediccion_compatible(X_features, y):
    """Crea y entrena el modelo de predicción"""
    caracteristicas_compatibles = [
        'intensidad_promedio', 'contraste', 'entropia', 'asimetria', 'curtosis',
        'densidad_bordes', 'magnitud_gradiente_promedio', 'hu_momento_1',
        'hu_momento_2', 'heterogeneidad'
    ]

    X_compatible = X_features[caracteristicas_compatibles]

    le = LabelEncoder()
    y_encoded = le.fit_transform(y)

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_compatible)

    model = RandomForestClassifier(
        n_estimators=200,
        max_depth=15,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=SEED
    )
    model.fit(X_scaled, y_encoded)

    return model, scaler, le, caracteristicas_compatibles

def predecir_imagen_tiroides(modelo, scaler, le, caracteristicas,
caracteristicas_compatibles):
    """Realiza predicción sobre una nueva imagen"""
    caracteristicas_ordenadas = [caracteristicas[col] for col in
caracteristicas_compatibles]
    caracteristicas_array = np.array(caracteristicas_ordenadas).reshape(1, -1)
```

```
    caracteristicas_esc = scaler.transform(caracteristicas_array)

    probabilidad = modelo.predict_proba(caracteristicas_esc)[0]
    prediccion = modelo.predict(caracteristicas_esc)[0]
    diagnostico = le.inverse_transform([prediccion])[0]
    confianza = max(probabilidad)

    return diagnostico, confianza, probabilidad
```

6. Sistema de Diagnóstico para Streamlit

```
python
def sistema_prediccion_doctor():
    """Sistema completo de diagnóstico para interfaz web"""
    # Cargar datos y entrenar modelo
    X, y, df_metadatos = cargar_dataset_completo_avanzado()
    X_features = df_metadatos.select_dtypes(include=[np.number])

    model, scaler, le, caracteristicas_compatibles =
crear_modelo_prediccion_compatible(
        X_features, y
    )

    # En Streamlit, procesar imagen subida
    archivo = st.file_uploader("Sube una imagen tiroidea",
                               type=["png", "jpg", "jpeg", "bmp", "tiff"])

    if archivo:
        # Procesar imagen y hacer predicción
        imagen = procesar_imagen_subida(archivo)
        caracteristicas = extraer_caracteristicas_avanzadas_completas(imagen)
        diagnostico, confianza, probabilidades = predecir_imagen_tiroides(
            model, scaler, le, caracteristicas, caracteristicas_compatibles
        )

        # Mostrar resultados
        generar_diagnostico_detallado(diagnostico, confianza, probabilidades,
                                      le, caracteristicas)
```

7. Utilidades

```
def es_archivo_imagen_avanzado(nombre):
  """Verifica si un archivo es una imagen válida"""
  extensiones_validas = ('.png', '.jpg', '.jpeg', '.bmp', '.tiff', '.tif', '.webp')
  return (nombre.lower().endswith(extensiones_validas) and
      not nombre.startswith('.') and
      os.path.isfile(nombre))

def crear_dataset_ejemplo():
  """Crea dataset de ejemplo para pruebas"""
  # Implementación para generar datos sintéticos
  pass
```

FLUJO DE PROCESAMIENTO PRINCIPAL:

1. **Carga** → cargar_dataset_completo_avanzado()
2. **Preprocesamiento** → cargar_y_preprocesar_imagen_avanzado()
3. **Extracción características** → extraer_caracteristicas_avanzadas_completas()
4. **Análisis EDA** → AdvancedEDA.perform_comprehensive_eda()
5. **Entrenamiento modelo** → crear_modelo_prediccion_compatible()
6. **Predicción** → predecir_imagen_tiroides()
7. **Diagnóstico** → generar_diagnostico_detallado()