Nombre del proyecto

INFORME TÉCNICO COMPLETO - SISTEMA DE DIAGNÓSTICO ASISTIDO POR IA PARA IMÁGENES TIROIDEAS

Profesor	Materia	Fecha de Inicio	Fecha de Finalización
ING. GLADYS VILLEGAS	PROYECTO INTEGRADOR	24/09/2025	30/09/2025

Resumen del Proyecto

El sistema de diagnóstico asistido desarrollado integra tecnologías avanzadas de inteligencia artificial con el objetivo de proporcionar una solución eficiente y precisa para el diagnóstico de patologías tiroideas a partir de imágenes médicas. Utilizando un enfoque basado en aprendizaje profundo, el sistema fue implementado en Google Colab, una plataforma de desarrollo que facilita la ejecución de código en la nube, y se apoya en la potencia de TensorFlow, una biblioteca líder en el campo de la inteligencia artificial.

El proceso comienza con el cargado y procesamiento de imágenes médicas de la glándula tiroides, almacenadas en una ruta específica en el entorno de trabajo (/content/drive/MyDrive/p_1_image). A partir de estas imágenes, el sistema aplica modelos de clasificación automática que permiten identificar posibles anormalidades o patologías, facilitando el trabajo de los médicos especialistas. Estos modelos han sido entrenados con grandes volúmenes de datos médicos para mejorar su precisión y confiabilidad.

Responsables del proyecto

Nombre	Rol	Tarea
Christian García	Estudiante	Informe Técnico-Fase de preparación y procesamiento de datos
Byron Piedra	Estudiante	Informe Técnico-Fase de preparación y procesamiento de datos

INFORME TÉCNICO COMPLETO - SISTEMA DE DIAGNÓSTICO ASISTIDO POR IA PARA IMÁGENES TIROIDEAS

1. RESUMEN EJECUTIVO

1.1 Contexto y Propósito

El sistema desarrollado representa una solución integral de inteligencia artificial para el diagnóstico asistido de patologías tiroideas mediante análisis de imágenes médicas. Implementado en Google Colab con TensorFlow, el sistema procesa imágenes localizadas en /content/drive/MyDrive/p_1_image y proporciona herramientas avanzadas de clasificación automática.

1.2 Arquitectura General

- **Framework Principal**: TensorFlow 2.12.0 con Keras
- **Backend de Procesamiento**: Google Colab con aceleración GPU T4
- **Almacenamiento**: Google Drive integrado
- **Método de Clasificación**: Redes Neuronales Convolucionales (CNN)

1.3 Capacidades del Sistema

- Clasificación multiclase: Malignant, Benign, Normal
- Preprocesamiento avanzado de imágenes
- Balanceo automático de datasets
- Análisis exploratorio completo (EDA)
- Diagnóstico asistido en tiempo real

2. METODOLOGÍA DE PROCESAMIENTO DE DATOS

2.1 Estructura del Dataset

El sistema está configurado para leer desde la ruta /content/drive/MyDrive/p_1_image con la siguiente estructura esperada:

text

```
p_1_image/
├── malignant/
│   ├── imagen1.jpg
│   ├── imagen2.png
```

```
│  └── ...
├── benign/
│   ├── imagen1.jpg
│   └── ...
└── normal/
    ├── imagen1.jpg
    └── ...
```

2.2 Parámetros de Configuración

python

```
BASE_PATH = "/content/drive/MyDrive/p_1_image"
CLASSES = ["malignant", "benign", "normal"]
IMG_SIZE = (299, 299)
BATCH_SIZE = 32
EPOCHS = 7
VALIDATION_SPLIT = 0.15
TEST_SPLIT = 0.15
MAX_IMAGES_PER_CLASS = 10000
```

2.3 Validación de Archivos

El sistema implementa verificación avanzada
mediante es_archivo_imagen_avanzado() que soporta múltiples formatos:

- PNG, JPG, JPEG, BMP, TIFF, TIF, WEBP
- Exclusión de archivos ocultos
- Verificación de integridad de archivos

3. PREPROCESAMIENTO AVANZADO DE IMÁGENES

3.1 Pipeline de Procesamiento

python

```
def cargar_y_preprocesar_imagen_avanzado(ruta, tamaño=IMG_SIZE):
```

Etapas del Pipeline:

1. **Apertura y Conversión**: Conversión automática a RGB
2. **Mejora de Calidad**:
    - Aumento de contraste (1.2x)
    - Mejora de nitidez (1.1x)
    - Ajuste de brillo (1.05x)
3. **Filtrado de Ruido**:
    - Filtro mediano 3x3
    - Suavizado avanzado
4. **Redimensionamiento**:
    - Thumbnail con LANCZOS
    - Resize final a 299x299
5. **Normalización**: Valores en rango [0, 1]

3.2 Extracción de Características

python

def extraer_caracteristicas_avanzadas_completas(arr):

Características Extraídas:

3.2.1 Características de Intensidad

- **Intensidad Promedio**: Valor medio de intensidad de píxeles
- **Contraste**: Desviación estándar de intensidades
- **Entropía**: Medida de complejidad textual

3.2.2 Momentos de Hu

- 7 momentos invariantes para análisis de forma
- Invariantes a rotación, escala y traslación

3.2.3 Análisis de Bordes

- **Detección Canny**: Bordes a 30-150 umbral
- **Densidad de Bordes**: Proporción de píxeles de borde
- **Magnitud de Gradiente**: Intensidad de transiciones

3.2.4 Estadísticas Avanzadas

- **Asimetría**: Simetría de distribución de intensidades
- **Curtosis**: Medida de "pico" de la distribución
- **Heterogeneidad**: Relación contraste/intensidad

4. ANÁLISIS EXPLORATORIO DE DATOS (EDA)

4.1 Análisis de Distribución

python

def realizar_eda_profesional_completo(df, X):

4.1.1 Métricas Generales

- Total de imágenes procesadas
- Dimensiones de imágenes (299x299x3)
- Distribución por clases
- Uso de memoria del dataset

4.1.2 Visualizaciones Implementadas

1. **Gráfico de Barras**: Distribución absoluta por clase
2. **Gráfico de Torta**: Proporciones porcentuales
3. **Histogramas**: Distribución de intensidad por clase
4. **Matriz de Correlación**: Relaciones entre características

4.2 Análisis de Balance

python

def balancear_dataset_avanzado(X, y, estrategia='auto'):

4.2.1 Estrategias de Balanceo

- **Auto**: Selección automática basada en ratio
- **Mixto Inteligente**: Percentil 75 como objetivo
- **Sobremuestreo Agresivo**: Máximo conteo como objetivo
- **Ninguno**: Para datasets balanceados

4.2.2 Criterios de Desbalance

- **Crítico**: Ratio > 5:1
- **Significativo**: Ratio 3:1 - 5:1

- **Moderado**: Ratio 2:1 - 3:1
- **Balanceado**: Ratio < 2:1

5. ARQUITECTURA DEL MODELO DE DEEP LEARNING

5.1 Modelo CNN Ultra-Robusto

python

def crear_modelo_ultra_robusto(input_shape, num_classes):

5.1.1 Capas de Entrada y Preprocesamiento

- **Input Layer**: 299x299x3
- **Rescaling**: Normalización 1./255
- **Data Augmentation**:
    - RandomFlip horizontal
    - RandomRotation ±10%

5.1.2 Bloques Convolucionales

python

 # Bloque 1

Conv2D(32, (3,3), activation='relu', padding='same')

MaxPooling2D((2,2))

BatchNormalization()

 # Bloque 2

Conv2D(64, (3,3), activation='relu', padding='same')

MaxPooling2D((2,2))

BatchNormalization()

 # Bloque 3

Conv2D(128, (3,3), activation='relu', padding='same')

MaxPooling2D((2,2))

BatchNormalization()

5.1.3 Capas Fully Connected

python

Flatten()

Dense(256, activation='relu')

Dropout(0.5)

Dense(128, activation='relu')

Dropout(0.3)

Dense(num_classes, activation='softmax')

5.2 Mecanismos de Regularización

5.2.1 Batch Normalization

- Normalización por lote después de cada conv
- Estabilización del entrenamiento
- Permite mayores learning rates

5.2.2 Dropout

- Capa 1: 50% de dropout
- Capa 2: 30% de dropout
- Prevención de overfitting

5.2.3 Early Stopping

python

```
EarlyStopping(
  monitor='val_loss',
  patience=8,
  restore_best_weights=True,
  min_delta=0.001
)
```

5.2.4 ReduceLROnPlateau

python

```
ReduceLROnPlateau(
```

```
    monitor='val_loss',
    factor=0.5,
    patience=4,
    min_lr=1e-7
)
```

6. ENTRENAMIENTO Y VALIDACIÓN

6.1 Preparación de Datos

python

```
# Encoding de labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_balanced)

# Pesos de clase
class_weights = class_weight.compute_class_weight(
    'balanced',
    classes=np.unique(y_encoded),
    y=y_encoded
)
```

6.2 División Estratificada

- **Entrenamiento**: 70% del dataset balanceado
- **Validación**: 15% del dataset balanceado
- **Test**: 15% del dataset balanceado

6.3 Entrenamiento Ultra-Seguro

python

```
def entrenamiento_ultra_seguro(model, X_train, y_train, X_val, y_val, epochs, batch_size):
```

6.3.1 Protocolo de 3 Niveles

1. **Intento Normal**: Entrenamiento estándar

2. **Recuperación**: Datos regenerados, LR reducido
3. **Emergencia**: Modelo mínimo, 2 épocas

6.3.2 Manejo de Errores

- Detección de InvalidArgumentError
- Regeneración de datos seguros
- Recompilación con parámetros conservadores

7. EVALUACIÓN DEL MODELO

7.1 Métricas de Evaluación

python

test_results = model.evaluate(X_test, y_test, verbose=0)

test_metrics = dict(zip(model.metrics_names, test_results))

7.1.1 Métricas Principales

- **Accuracy**: Precisión general
- **Precision**: Valor predictivo positivo
- **Recall**: Sensibilidad
- **Loss**: Función de pérdida

7.2 Matriz de Confusión

- Visualización con seaborn heatmap
- Anotaciones numéricas
- Colormap Blues para mejor legibilidad

7.3 Reporte de Clasificación

python

print(classification_report(y_test, y_pred_classes, target_names=label_encoder.classes_))