Informe Técnico: Sistema de Reconocimiento Óptico de Caracteres (OCR) con Edge AI en Flutter

Equipo de Desarrollo

24 de Junio de 2025

Contents

1 Introducción

Este informe detalla el desarrollo de un sistema de reconocimiento óptico de caracteres (OCR) basado en técnicas de Deep Learning optimizadas para Edge AI. El sistema permite extraer texto de imágenes capturadas por cámara o seleccionadas desde la galería, ejecutando la inferencia completamente en dispositivos móviles sin conexión a internet. La implementación se realiza a través de una aplicación multiplataforma desarrollada en Flutter, asegurando compatibilidad con Android e iOS.

El proyecto abarca desde la preparación del dataset y el entrenamiento de modelos hasta la optimización para dispositivos de borde y la integración en la app. Se utilizan técnicas avanzadas como destilación de conocimiento, poda (pruning) y cuantización para garantizar un rendimiento eficiente en hardware limitado.

2 Pipeline de Machine Learning y Edge AI

2.1 Dataset y Preprocesamiento

El dataset consta de imágenes de caracteres alfanuméricos (0-9, A-Z), organizadas por clase en carpetas separadas. Cada imagen se preprocesa para optimizar su uso en el entrenamiento del modelo. Los pasos incluyen:

- Conversión a escala de grises.
- Redimensionamiento a 64x64 píxeles.
- Normalización de valores de píxeles al rango [0, 1].

A continuación, se muestra un ejemplo de código en Python para el preprocesamiento:

```python
import cv2
import numpy as np

img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
img_resized = cv2.resize(img, (64, 64))
img_normalized = img_resized.astype(np.float32) / 255.0
img_final = np.expand_dims(img_normalized, axis=-1)
```

Listing 1: Preprocesamiento de imágenes en Python

2.2 Modelos y Técnicas de Optimización

Se desarrollaron dos modelos principales: un modelo profesor y un modelo estudiante, optimizados para Edge AI.

2.2.1 Modelo Profesor

El modelo profesor, basado en EfficientNetB0, se entrena para alcanzar la máxima precisión. Aunque robusto, su tamaño y complejidad lo hacen inadecuado para dispositivos de borde sin optimización.

2.2.2 *Modelo Estudiante*

El modelo estudiante es una red neuronal convolucional (CNN) ligera, diseñada específicamente para dispositivos móviles. Tiene menos parámetros y capas, priorizando la velocidad de inferencia sin sacrificar demasiada precisión.

2.2.3 *Destilación de Conocimiento*

La destilación de conocimiento permite transferir el aprendizaje del modelo profesor al modelo estudiante. Esto mejora la precisión del modelo ligero, utilizando las predicciones del profesor como guía durante el entrenamiento.

2.2.4 *Poda (Pruning)*

La poda elimina conexiones y pesos poco relevantes del modelo estudiante, reduciendo su tamaño y acelerando la inferencia. Se emplea una estrategia de poda gradual, como se muestra en el siguiente código:

```
import tensorflow_model_optimization as tfmot

pruning_params = {
    'pruning_schedule': tfmot.sparsity.keras.PolynomialDecay(
        initial_sparsity=0.30, final_sparsity=0.70, begin_step=0,
           end_step=1000
    )
}
model_for_pruning = tfmot.sparsity.keras.prune_low_magnitude(model,
   **pruning_params)
```

Listing 2: Aplicación de poda en TensorFlow

2.2.5 *Cuantización (Quantization)*

El modelo podado se convierte a formato TFLite con cuantización int8, reduciendo aún más su tamaño y optimizando la inferencia en dispositivos de borde. Un ejemplo de conversión a TFLite:

```
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data_gen
converter.target_spec.supported_ops =
   [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8
quantized_model = converter.convert()
```

Listing 3: Conversión a TFLite con cuantización

2.3 Exportación y Evaluación

El modelo final se exporta como `ocr_model_quantized.tflite`. Se evalúan métricas clave como precisión, tamaño del modelo y velocidad de inferencia en cada etapa (profesor, estudiante, podado, cuantizado).

3 Integración en la Aplicación Flutter

3.1 Estructura de la Aplicación

La aplicación, nombrada `ocr_edge_app`, está desarrollada en Flutter para garantizar compatibilidad multiplataforma. Sus componentes principales son:

- **Pantalla principal:** Navegación a las funcionalidades de OCR por cámara, OCR por galería e historial.
- **OCR por cámara:** Captura de imagen en tiempo real, preprocesamiento e inferencia.
- **OCR por galería:** Selección de imagen desde el dispositivo, preprocesamiento e inferencia.
- **Historial:** Registro y visualización de resultados previos.

3.2 Inferencia con TFLite en Flutter

La inferencia se realiza utilizando el paquete `tflite_flutter`. El modelo TFLite se carga en la app, y las imágenes se preprocesan en Dart antes de la inferencia. A continuación, se muestra un ejemplo de código:

```
import 'package:tflite_flutter/tflite_flutter.dart';
import 'package:image/image.dart' as img;

// Cargar el modelo
_interpreter = await
   Interpreter.fromAsset('model_hybrid_quantized.tflite');

// Preprocesar imagen
final grayscale = img.grayscale(image);
final resized = img.copyResize(grayscale, width: 64, height: 64);
var inputArray = Float32List(1 * 64 * 64 * 1);
var inputIndex = 0;
for (var y = 0; y < 64; y++) {
  for (var x = 0; x < 64; x++) {
    inputArray[inputIndex++] = img.getRed(resized.getPixel(x,
       y)).toDouble();
  }
}

// Inferencia
var outputArray = Float32List(1 * 36);
_interpreter!.resizeInputTensor(0, [1, 64, 64, 1]);
_interpreter!.allocateTensors();
```

```
_interpreter!.run(inputArray, outputArray);

// Obtener resultado
var maxIndex = outputArray.indexWhere((v) => v ==
   outputArray.reduce(max));
final recognizedChar = ocrClasses[maxIndex];
```

Listing 4: Carga e inferencia en Flutter

3.3 Ventajas de Edge AI

La implementación en Edge AI ofrece:

- **Procesamiento local:** Sin dependencia de internet, garantizando privacidad y baja latencia.
- **Optimización:** Modelos pequeños y rápidos gracias a destilación, poda y cuantización.
- **Experiencia de usuario:** Respuestas inmediatas y una interfaz intuitiva.

4 Pasos del Proyecto

El desarrollo del proyecto sigue estos pasos:

1. Preparación y exploración del dataset.
2. Entrenamiento del modelo profesor.
3. Entrenamiento del modelo estudiante con destilación de conocimiento.
4. Aplicación de poda al modelo estudiante.
5. Cuantización del modelo podado y exportación a TFLite.
6. Integración del modelo TFLite en la app Flutter.
7. Implementación de la inferencia y la interfaz de usuario.
8. Evaluación de precisión, tamaño y velocidad.

5 Archivos y Referencias Clave

- **Notebook de entrenamiento:** `python/modelocr.ipynb`
- **Modelo TFLite:** `python/ocr_model_quantized.tflite`
- **App Flutter:** Carpeta `ocr_edge_app/`
  - `lib/ocr_screen.dart`
  - `lib/gallery_ocr_screen.dart`
  - `lib/main.dart`

6 Conclusiones

Este proyecto demuestra la viabilidad de implementar un sistema OCR eficiente en dispositivos de borde utilizando Edge AI. Las técnicas de optimización aplicadas (destilación, poda y cuantización) permiten ejecutar modelos de Deep Learning en hardware limitado, manteniendo una alta precisión y velocidad. La app Flutter ofrece una experiencia de usuario fluida y multiplataforma, destacando las ventajas de procesar datos localmente.

El sistema es escalable y adaptable a otros problemas de visión por computadora, consolidando un enfoque moderno para aplicaciones de inteligencia artificial en dispositivos móviles.