Clase 169 — TF Lite (mobile/embedded)

Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Deploying a Model to a Mobile or Embedded Device. Duración estimada: 55 min.

Clase 169 — TF Lite (mobile/embedded)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Deploying a Model to a Mobile or Embedded Device. Duración estimada: 55 min.*

Objetivo

Convertir y desplegar un modelo a TensorFlow Lite (LiteRT) — runtime optimizado para móviles (Android/iOS), IoT y embedded (Raspberry Pi, microcontroladores). Aplicar quantization (int8) para reducir tamaño 4× y acelerar 2-4× en CPU móvil.

> *Nota: TF Lite fue renombrado a LiteRT en 2024 (mismo proyecto, nuevo branding bajo el paraguas de "AI Edge").*

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Convertir SavedModel a .tflite: converter = tf.lite.TFLiteConverter.from_saved_model('servable/1/'); tflite_model = converter.convert().
- Aplicar quantization post-training (dynamic range, int8 full).
- Cargar y ejecutar inference con tf.lite.Interpreter.
- Conocer alternativas modernas: ONNX Runtime Mobile, CoreML (iOS), NNAPI (Android).
- Reconocer trade-off accuracy vs tamaño/velocidad.

Temas

- TF Lite vs TF: subset de ops, runtime minimal.
- Conversión SavedModel → tflite.
- Quantization types: dynamic range (weights int8), int8 full (weights + activations), float16.
- tf.lite.Interpreter API.
- Mobile delegates: NNAPI, GPU, CoreML.

Definiciones y características

- TF Lite: runtime para edge devices.
- Quantization: convertir float32 → int8/float16 → modelo más chico y rápido.
- Calibration dataset: ~100 ejemplos para calibrar quantization de activaciones.
- Delegate: hardware-specific accelerator (NNAPI en Android, CoreML en iOS, Edge TPU).
- .tflite: archivo binario compacto.

Dataset / recursos

- Modelo Fashion-MNIST.
- Librerías: tensorflow.

Ejercicios

1. Convert: TFLiteConverter.from_saved_model(...).convert() → guardar .tflite. Comparar tamaño vs SavedModel.
2. Inference: cargar .tflite y hacer predict con Interpreter. Verificar misma predicción que el modelo original.
3. Dynamic range quant: converter.optimizations = [tf.lite.Optimize.DEFAULT]. Tamaño 4× menor.
4. Full int8: definir representative_dataset y converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]. Comparar tamaño y accuracy.
5. Latencia: medir tiempo de inference en CPU laptop. Comparar versión float32 vs int8.

Homework verificable

Pipeline de Fashion-MNIST a TF Lite:

1. Convertir + 3 quantizations (none, dynamic, int8 full).
2. Para cada uno: tamaño + accuracy en test + latencia.
3. Reportar tabla comparativa.

Criterio de aceptación: int8 reduce tamaño ~4× con < 1 pp accuracy loss; latencia 2-3× más rápida.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
op not supported al convertir	Algunas ops Keras no son TFLite-compatible
int8 full quant con representative dataset	Mala calibración. Fix: ≥ 100 ejemplos dive
Tamaño del .tflite no baja con quant	El modelo era pequeño. Quantization brilla
Inference más lenta de lo esperado	Sin delegate. Fix: NNAPI (Android), CoreML
Predicciones distintas float vs int8	Esperable; verificar accuracy en test set.

Preguntas frecuentes

¿TFLite o ONNX Runtime Mobile?

TFLite si entrenas en TF; ONNX para portabilidad multi-framework. Funcionalmente comparable.

¿iOS — TFLite o CoreML?

CoreML mejor integración con iOS (Neural Engine de Apple). Convertir TFLite → CoreML con coremltools.

¿Microcontroladores ARM Cortex-M?

TF Lite Micro (TFLM) o MicroPython TF — para inference en MCU con KB de RAM. Modelos diminutos (< 100 KB).

¿Quantization durante training?

QAT (Quantization-Aware Training) — entrenar simulando los efectos de int8. Mejor accuracy que post-training quant.

¿Cuándo NO desplegar a mobile?

Modelos grandes (>100 MB) o que requieren GPU server (LLMs grandes). Para LLMs en mobile, ver MLC LLM o llama.cpp con quantization GGUF.

Referencias

- Géron, cap. 19 — Deploying a Model to a Mobile or Embedded Device.
- TensorFlow Lite (LiteRT) docs.
- ONNX Runtime Mobile.
- CoreML Tools.

Siguiente clase

Clase 170 — TensorFlow.js (navegador)

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb