Clase 127 — TensorFlow Datasets (TFDS)

Parte: 2 — Deep Learning · Fuente: Géron, cap. 13 § The TensorFlow Datasets (TFDS) Project. Duración estimada: 40 min.

Clase 127 — TensorFlow Datasets (TFDS)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 13 § The TensorFlow Datasets (TFDS) Project. Duración estimada: 40 min.*

Objetivo

Conocer TFDS —catálogo de datasets prearmados (CIFAR, ImageNet, IMDB, COCO, MNIST, GLUE, etc.)— y la alternativa moderna Hugging Face datasets (estándar en NLP/LLMs). Cargar datasets de prueba, hacer splits, y entender por qué TFDS es práctico para benchmarks reproducibles.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Listar datasets disponibles con tfds.list_builders().
- Cargar con tfds.load('cifar10', split=['train', 'test'], as_supervised=True).
- Hacer splits custom con la slicing API: 'train[:80%]', 'train[80%:]'.
- Reconocer cuando usar tfds vs huggingface_hub.datasets.

Temas

- Catálogo TFDS: 200+ datasets, descarga automática + cache.
- as_supervised=True → tuplas (x, y).
- Splits: 'train[:80%]', 'train[-20%:]', 'all'.
- dataset.info con metadata (shape, num_classes, etc.).
- Hugging Face datasets: estándar moderno multi-framework.

Definiciones y características

- TFDS: librería de Google con datasets pre-procesados en formato TFRecord.
- as_supervised: si True, devuelve (input, label). Si False, devuelve dict con todas las features.
- Slicing API: notación tipo Python sobre los splits.
- Hugging Face datasets: equivalente moderno, basado en Arrow, soporta PyTorch/TF/JAX, comunidad enorme.

Dataset / recursos

- CIFAR-10 vía TFDS.
- IMDB vía HF datasets.
- Librerías: tensorflow-datasets (pip install tensorflow-datasets), opcional datasets (HF).

Ejercicios

1. Listar: tfds.list_builders() → primeros 20 datasets.
2. CIFAR-10: (ds_train, ds_test), info = tfds.load('cifar10', split=['train', 'test'], as_supervised=True,

with_info=True). Imprimir info.

3. Slicing: cargar train[:90%] + train[90%:] como train/val split.
4. Pipeline: ds_train.map(preprocess).cache().shuffle(1024).batch(32).prefetch(AUTOTUNE).
5. HF datasets: from datasets import load_dataset; ds = load_dataset('imdb'). Inspeccionar; convertir a tf.data con ds.to_tf_dataset(...).

Homework verificable

Entrenar un MLP simple en CIFAR-10 cargado vía TFDS:

1. Cargar con tfds.load(..., as_supervised=True).
2. Pipeline con preprocessing, batch, prefetch.
3. MLP [1024, 512, 256, 10] con BN + Dropout.
4. Reportar accuracy en test.

Criterio de aceptación: accuracy en test ≥ 0.45 (MLP no es ideal para CIFAR — CNN gana — pero debe llegar a 45 %; clase 113+ lo mejora con CNN).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Primera carga muy lenta	TFDS descarga + procesa. Cachea en ~/tenso
OutOfRangeError al iterar	Te quedaste sin batches sin pasar .repeat(
Olvidar as_supervised=True	El dataset devuelve dicts, modelo espera t
tfds.load(split='train+test')	Concatena ambos splits. Para train/val: us
Disco lleno por datasets grandes (ImageNet	TFDS cachea todo. Fix: borrar ~/tensorflow

Preguntas frecuentes

¿TFDS o HF datasets?

Para benchmarks tradicionales (CIFAR, IMDB, MNIST) ambos sirven. Para NLP moderno (text generation, instruction datasets), HF. Para TF nativo + GCP/Vertex AI, TFDS.

¿TFDS funciona con PyTorch?

Sí, con tfds.as_numpy(...) o tfds.load(..., builder_kwargs={'as_dataset_kwargs': ...}). Pero HF es más natural.

¿Datasets propios cargo cómo?

Para datasets locales no listados: tfds.folder_dataset.ImageFolder('/path'), o construir un DatasetBuilder custom. En HF: load_dataset('csv', data_files='...').

¿Datasets de imagen grandes (ImageNet)?

TFDS los maneja bien (shardea, cache). Para TPU + Vertex AI, ya está optimizado.

¿Versionado de datasets?

TFDS versiona por defecto ('cifar10:3.0.2'). HF también. Es importante reportarlo en papers/reports.

Referencias

- Géron, cap. 13 — The TensorFlow Datasets (TFDS) Project.
- TFDS catalog.
- Hugging Face datasets.

Siguiente clase

Clase 128 — Capas convolucionales, filtros, feature maps

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb