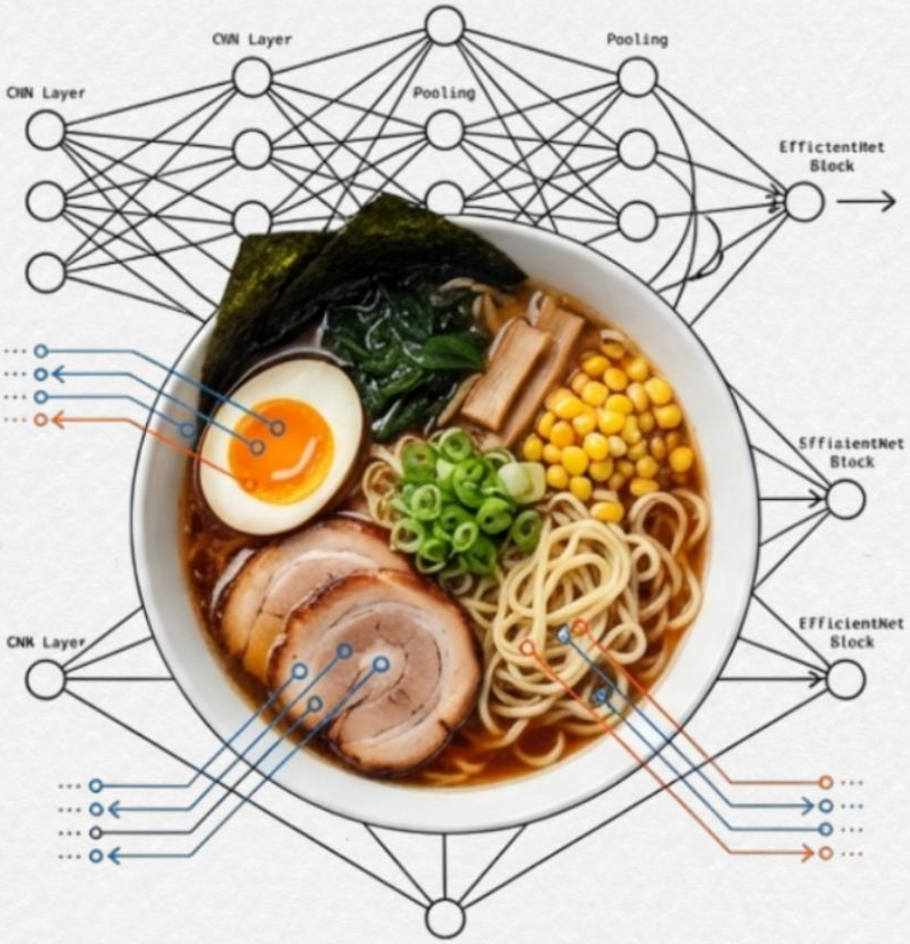
PROYECTO FINAL

ANALIZADOR DE CALORÍAS



GERARDO BLÁZQUEZ MORENO

Índice

Gerardo Blázquez Moreno

1. Introducción y Objetivo

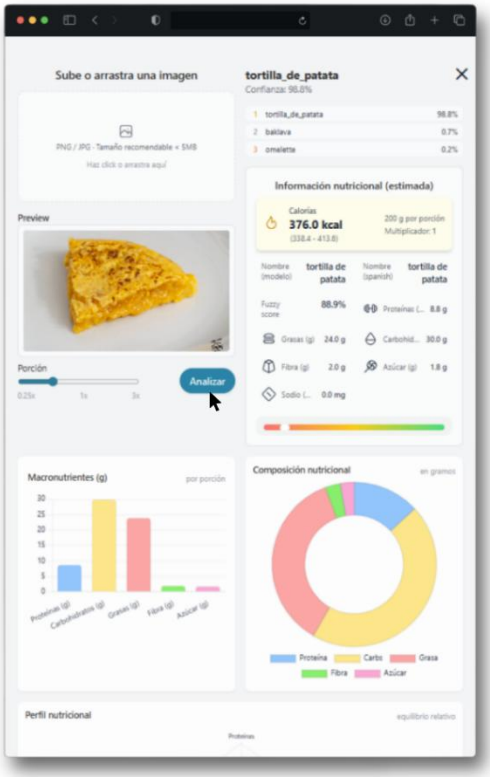
El desarrollo de un sistema de Inteligencia Artificial especializado en la clasificación de imágenes de alimentos y no-alimentos. Basado en **redes neuronales convolucionales (CNN)**, el sistema emplea una arquitectura en cascada diseñada para garantizar **precisión**, **robustez** y **escalabilidad**.

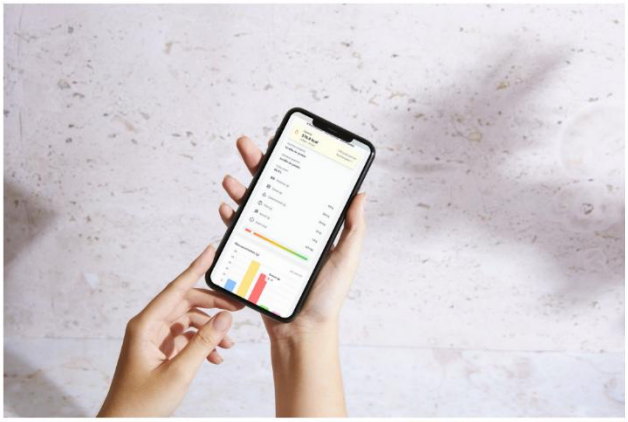
La solución se estructura en **tres niveles de clasificación**:

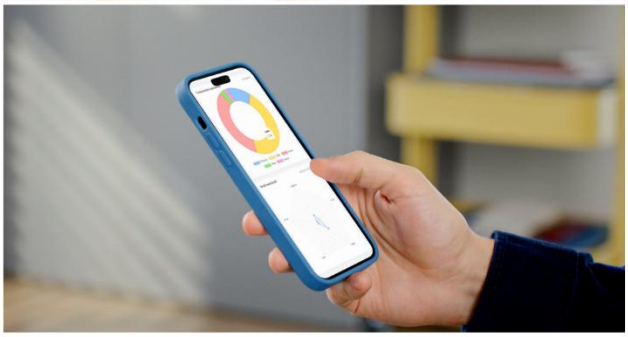
1. Un filtro binario inicial ((**0**) Comida / (**1**) No-Comida).
2. Seguido de modelos especializados que identifican 121 clases de alimentos.
3. 22 categorías de no-alimentos.

Utilizando el *backbone* **EfficientNet-B0**, este proyecto no solo busca la identificación visual, sino que sienta las bases para la integración de análisis nutricionales añadiendo más clases que el clásico dataset Food-101, diferenciación de comida y no comida y servicios de clasificación automática en plataformas móviles y web.

Actualmente dispone de un despliegue a modo de "*demo*" en mi portfolio gcodev.es







Implementa una interfaz de usuario intuitiva, sencilla y ágil, con la cual el usuario en cuestión de segundos tirne toda la información nutricional de la comida que vaya a realizar.

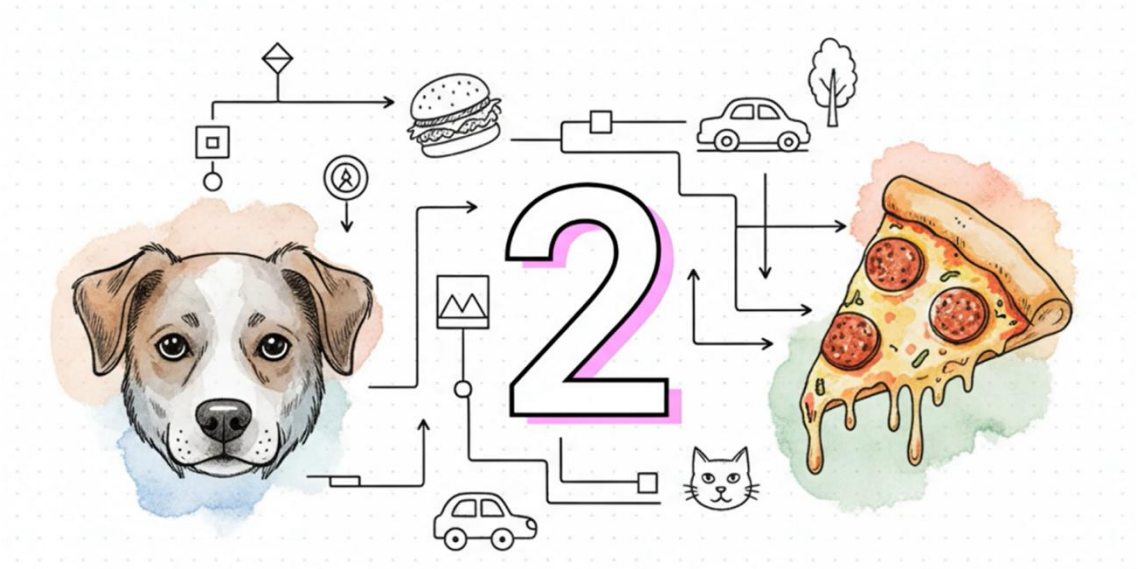
1.1 Arquitectura del Sistema en Cascada

El sistema opera bajo una tubería modular que permite procesar la información de forma secuencial, optimizando el rendimiento y reduciendo errores en las etapas finales.

1.2 Clasificación Binaria (Filtro Principal)

El primer modelo determina si una imagen contiene comida o no. Este paso es crítico para:

- Evitar que el sistema intente clasificar como alimento un objeto que no lo es.
- Mejorar la robustez del sistema en entornos reales.
- Reducir el ruido en las etapas posteriores de clasificación multiclase.



1.3 Clasificación de Alimentos (121 Clases)

Si el filtro inicial identifica comida, se activa un modelo entrenado con el dataset **Food-101** (ajustado para este proyecto).

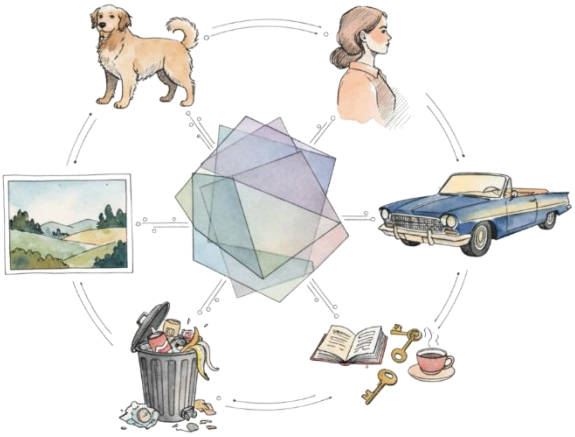
- **Capacidad:** Identifica el tipo exacto de alimento entre 121 categorías.

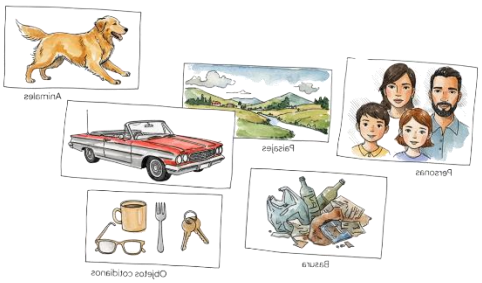
- **Utilidad:** Funciona como base para asociar métricas nutricionales (calorías, macronutrientes).

1.4 Clasificación de No-Alimentos (22 Clases)

Si la imagen es descartada como comida, el sistema la clasifica en categorías contextuales para comprender mejor el entorno visual. Las clases incluyen:

- Personas y animales.
- Objetos cotidianos y vehículos.
- Paisajes y basura.



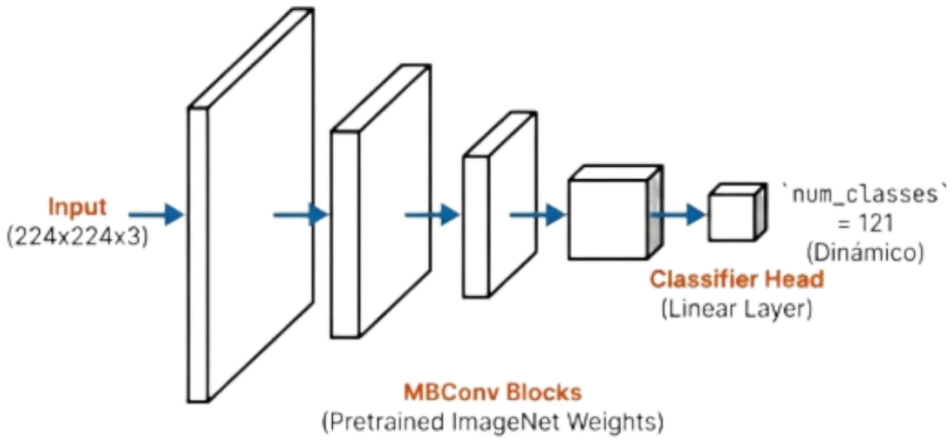


2. Metodología Técnica y Entrenamiento

El desarrollo técnico se apoya en herramientas de vanguardia para visión por computador y procesamiento de datos.

Especificaciones del Modelo

- **Arquitectura Base (Backbone):** efficientnet_b0 (vía librería timm).

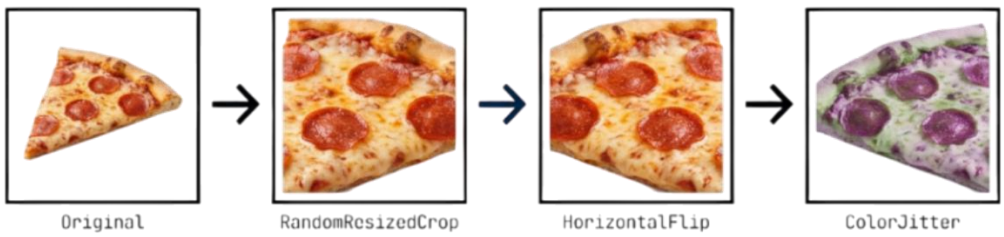


- **Optimización:** Algoritmo **AdamW**.
- **Función de Pérdida:** CrossEntropyLoss con suavizado de etiquetas (*label smoothing*) de 0.1.
- **Aceleración:** Uso de **AMP (Automatic Mixed Precision)** para un entrenamiento más rápido y estable.

Preprocesamiento y Aumentos de Datos

Se utiliza la librería **Albumentations** para mejorar la capacidad de generalización del modelo mediante:

- RandomResizedCrop
- HorizontalFlip
- ColorJitter
- **Normalización:** Media (0.485, 0.456, 0.406) y desviación estándar (0.229, 0.224, 0.225).



Manejo de Datos

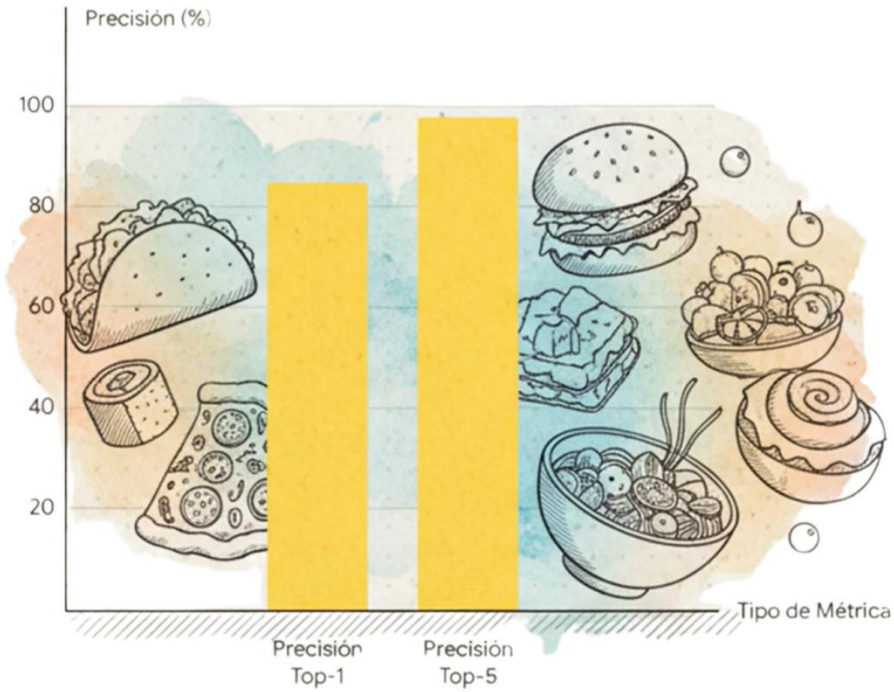
El sistema implementa WeightedRandomSampler para corregir posibles desequilibrios entre las clases del dataset, asegurando que el modelo aprenda equitativamente de todas las categorías.

3. Evaluación y Métricas de Rendimiento

El éxito del sistema se mide a través de diversos indicadores estándar en visión por computador:

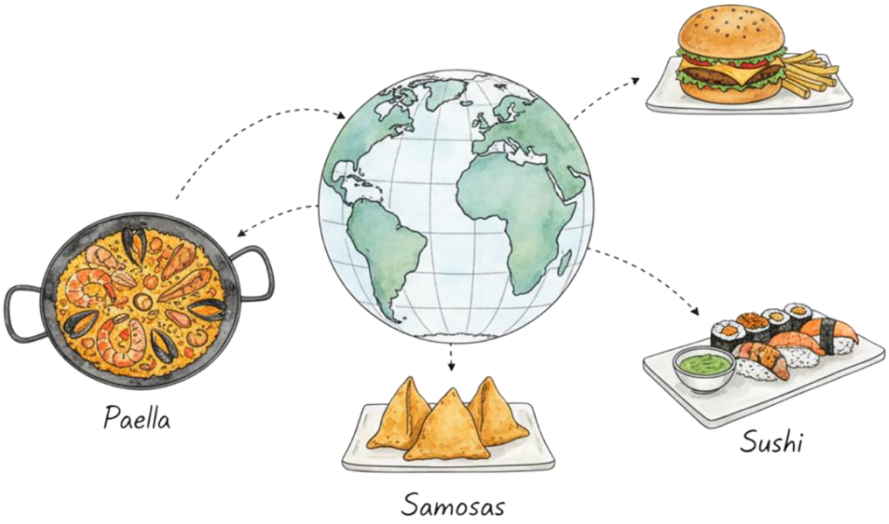
Métrica	Descripción y Utilidad
Top-1 Accuracy	Probabilidad de acierto en la etiqueta con mayor confianza; crítico para decisiones automáticas.
Top-3 / Top-5 Accuracy	Útiles para interfaces que ofrecen sugerencias o toleran cierto margen de error.
F1-Score por Clase	Permite identificar clases individuales con rendimiento débil para su optimización.
Matriz de Confusión	Visualiza los errores de clasificación entre clases similares.

Además, el sistema genera visualizaciones de la distribución de confianza y barras de *recall* por clase para un análisis incisivo de su comportamiento.



4. Implementación y Reproducibilidad

El proyecto está diseñado para ser reproducible y fácilmente integrable en entornos productivos.



4.1 Organización del Repositorio

- **train.py**: Script principal compatible con modos binario y multiclase.

  Estrategia del entrenamiento:

Parámetro	Valor
Optimizador	AdamW (lr=3e-4)
Loss Function	CrossEntropy (Label Smoothing 0.1)
Batch Size	16–32
Epochs	10–30

- **inference_cascade.py**: Lógica de ejecución de la tubería completa (cascada).
- **evaluate.py:** Herramientas de análisis de métricas.
- **utils.py**: Funciones auxiliares y de carga de datos.

4.2 Consideraciones para el Despliegue

Para asegurar la consistencia en producción, es obligatorio guardar los archivos classes_food.txt y classes_binary.txt. Esto garantiza que la inferencia utilice exactamente el mismo orden de etiquetas empleado durante el entrenamiento.

Solución de Problemas Comunes

- **Error de tamaño en el clasificador:** Indica un desajuste entre las clases del punto de control (*checkpoint*) y el modelo actual. Se resuelve cargando correctamente el archivo de clases.
- **Bloqueos en DataLoader (Colab):** Se recomienda configurar num_workers=0.
- **Imágenes oscuras:** Debido a la normalización de tensores; requieren desnormalización para su visualización correcta.

5. Visiones Futuras y Consideraciones Éticas

El sistema está concebido como una plataforma extensible con múltiples vías de mejora:

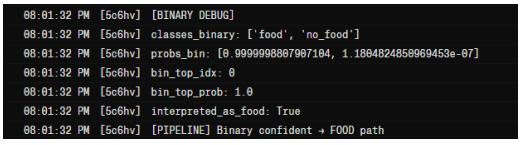
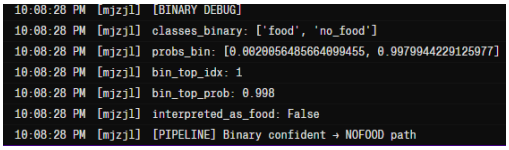
1. **Módulo Nutricional:** Implementación de una base de datos que asocie cada clase de alimento con calorías, proteínas, carbohidratos y grasas, permitiendo un cálculo estimado basado en la imagen.
2. **Interpretabilidad:** Uso de **Grad-CAM** para visualizar mapas de activación y entender en qué partes de la imagen se enfoca el modelo para decidir.
3. **Optimización para Producción:** Conversión de modelos a formatos **TorchScript** o **ONNX** para una inferencia de alta velocidad.
4. **Ética y Sesgos:** Se reconoce que el dataset (Food-101) puede tener una representación cultural limitada. El sistema no debe utilizarse para diagnósticos médicos sin supervisión humana y bases científicas rigurosas.

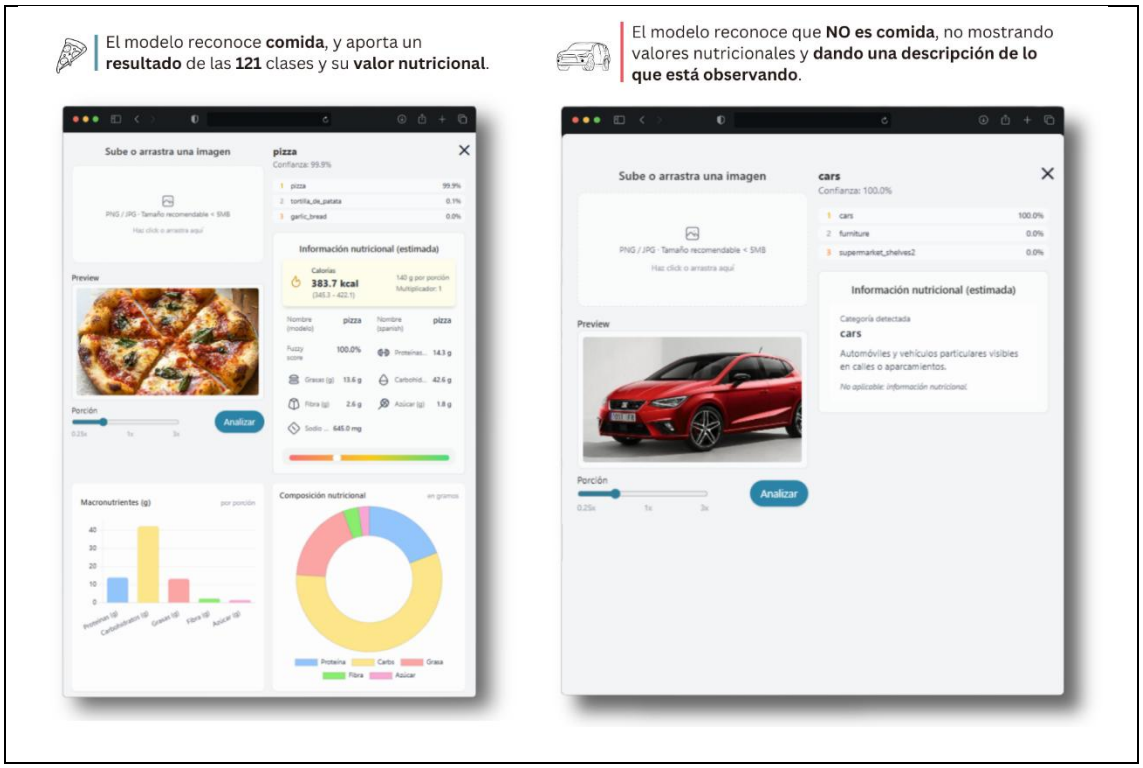
6. Despliegue Real

El sistema de clasificación de alimentos y no-alimentos está actualmente desplegado como demo en mi portfolio personal (gcodev.es). Este despliegue permite demostrar el potencial del modelo experimentando directamente con la tecnología desarrollada, subiendo o seleccionando imágenes para ver cómo el modelo identifica visualmente si se trata de un alimento o de un objeto/escena no relacionada con comida.

Todas las imágenes que se muestran en la demo forman parte de mi portfolio y fueron seleccionadas para ilustrar el comportamiento del sistema en contextos reales. Esto permite demostrar no solo la precisión del modelo, sino también su robustez frente a imágenes cotidianas, variadas y con diferentes condiciones de luz, ángulo o entorno.

Ejemplos del despliegue

Comida (Pizza)	No-Comida (Coche)
En la demo, al seleccionar la imagen de una pizza margherita tomada de mi portfolio, el sistema realiza la clasificación en cascada: 1. Determina que es un alimento 2. Luego identifica la clase específica dentro de las 121 categorías de alimentos. Además, la interfaz muestra una estimación de información nutricional, indicando calorías, proteínas, carbohidratos y grasas por porción. Esto permite al usuario comprender rápidamente los valores nutricionales asociados a la comida que aparece en la imagen. **Se muestran los logs del servidor**  08:01:32 PM [5c6hv] [BINARY DEBUG] 08:01:32 PM [5c6hv] classes_binary: ['food', 'no_food'] 08:01:32 PM [5c6hv] probs_bin: [0.9999998807907104, 1.1804824850969453e-07] 08:01:32 PM [5c6hv] bin_top_idx: 0 08:01:32 PM [5c6hv] bin_top_prob: 1.0 08:01:32 PM [5c6hv] interpreted_as_food: True 08:01:32 PM [5c6hv] [PIPELINE] Binary confident → FOOD path	Al seleccionar la imagen de un **Coche**, el **filtro binario** reconoce que **no** se trata de un **alimento** y activa el clasificador de **no-alimentos**, asignando la categoría correspondiente entre las **22** disponibles. La interfaz indica claramente que **no** hay cálculo **nutricional** disponible y proporciona la clase identificada ("*Cars*"), mostrando una descripción "*Automóviles y vehículos particulares visibles en calles o aparcamientos*." Demostrando cómo el sistema maneja correctamente imágenes fuera del dominio alimentario. **Se muestran los logs del servidor**  10:08:28 PM [mjzjl] [BINARY DEBUG] 10:08:28 PM [mjzjl] classes_binary: ['food', 'no_food'] 10:08:28 PM [mjzjl] probs_bin: [0.0020056485664099455, 0.9979944229125977] 10:08:28 PM [mjzjl] bin_top_idx: 1 10:08:28 PM [mjzjl] bin_top_prob: 0.998 10:08:28 PM [mjzjl] interpreted_as_food: False 10:08:28 PM [mjzjl] [PIPELINE] Binary confident → NOFOOD path



Interfaz de usuario

La demo ofrece una experiencia **intuitiva** y **ágil**:

- Selección de **imágenes** desde la **galería** del **portfolio**.
- Resultados visuales inmediatos con la clase **identificada** y su **confianza**.
- Información nutricional para **alimentos** y **categorización contextual** para no-alimentos.

Este despliegue real demuestra la aplicabilidad del sistema en escenarios cotidianos y sirve como base para futuras integraciones en plataformas móviles o web, donde la identificación rápida de alimentos y objetos pueda ser útil para análisis nutricionales, etiquetado automático o sistemas de recomendación visual.

Infografía ilustrativa del Proyecto

Demostración del código del Notebook de Google Collab

Objetivo

El objetivo de este proyecto es desarrollar un **sistema de Inteligencia Artificial basado en redes neuronales convolucionales (CNN)** de **identificar y clasificar imágenes de comida y no comida** de forma precisa y escalable.

El sistema se construye siguiendo una **arquitectura en cascada**, donde distintos modelos especializados se encargan de resolver cada etapa del problema.

Clasificación binaria (Food vs No-Food)

Este módulo entrena un primer modelo encargado de determinar si una imagen contiene **comida** o **no comida**, actuando como filtro inicial del sistema.

Qué hace

1. Analiza la imagen de entrada.
2. Predice si pertenece a la categoría food o no_food.
3. Decide qué modelo especializado debe ejecutarse a continuación.

Ventajas

- Reduce errores en la clasificación final.
- Evita clasificar imágenes no relacionadas como comida.
- Mejora la robustez del sistema en entornos reales.

Clasificación de alimentos (121 clases)

Si la imagen es clasificada como food, se aplica un segundo modelo entrenado para reconocer **121 tipos distintos de alimentos**, utilizando el dataset **Food-101 **(adaptado y ampliado).

Qué hace

1. Recibe únicamente imágenes previamente clasificadas como comida.
2. Predice la clase concreta del alimento.
3. Devuelve una probabilidad asociada a cada clase.

Aplicaciones

- Identificación precisa del tipo de alimento.
- Asociación con información nutricional (calorías, macronutrientes, etc.).
- Sistemas de control dietético y recomendación alimentaria.

Clasificación de no-alimentos (22 clases)

Si la imagen es clasificada como `no_food`, se utiliza un modelo específico entrenado para identificar **22 categorías diferentes de no comida.**

Clases incluidas

- Personas
- Animales
- Basura
- Objetos cotidianos
- Paisajes
- Vehiculos, entre otros

Objetivo

- No solo descartar imágenes de comida, sino comprender el contexto visual general de la imagen.

Evaluación y métricas

El rendimiento del sistema se evalúa utilizando métricas estándar en visión por computador.

Métricas utilizadas

- Top-1 Accuracy
- Top-3 Accuracy
- Top-5 Accuracy
- Precision, Recall y F1-score por clase
- Matrices de confusión (globales y por subconjuntos de clases)

Estas métricas permiten evaluar tanto el rendimiento global como el comportamiento por clase.

Resultado final

El resultado es un sistema **modular, reutilizable y extensible** que puede integrarse en:

- Aplicaciones móviles
- Plataformas de análisis de imágenes
- Sistemas de recomendación nutricional
- Servicios automáticos de clasificación visual

Instalación de dependencias

Hay que instalar las librerias necesarias para entrenamiento e inferencia y ejecutar esta celda en una celda de notebook o en una terminal.

Paquetes principales

- `timm==0.9.2` —> modelos EfficientNet y otros.
- `torchmetrics`, `albumentations==1.3.0`, `opencv-python-headless` —> utilidades para métricas y aumentos.
- `scikit-learn` —> utilidades (split, métricas).
- `fastai==1.0.61` —> opcional

> Nota: usar GPU para entrenamiento; en Colab el runtime debe ser GPU para su correcto funcionamiento.

```
# Ejecutar en una celda de terminal o notebook con !
!pip install -q timm==0.9.2 torchmetrics albumentations==1.3.0 opencv-python-headless
# opcional para fastai approach
!pip install -q fastai==1.0.61

!pip install -q scikit-learn
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 68.5/68.5 kB 2.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 43.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 123.5/123.5 kB 10.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 983.2/983.2 kB 48.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 239.2/239.2 kB 8.0 MB/s eta 0:00:00
  Building wheel for nvidia-ml-py3 (setup.py) ... done
```

Montar Google Drive y copiar dataset

Esta celda monta Google Drive y valida la estructura del dataset `Food-101`. Explica lo que hace el script:

Qué hace

1. Monta Drive en `/content/drive`.
2. Define rutas: `DRIVE_DATA_DIR = "/content/drive/MyDrive/Food-101"` y `LOCAL_DATA_DIR = "/content/Food-101"`.
3. `dataset_ok(base)` verifica que existan `train` y `val` y un número razonable de clases.
4. Si el dataset es válido en Drive y no existe en local, lo copia a `/content` (más rápido para entrenamiento en Colab).

Recomendaciones

- Copiar a disco local (`copytree`) es recomendable en Colab para evitar I/O lento sobre Drive.
- Si la comprobación falla, revisa que las carpetas `train` y `val` estén correctamente estructuradas.

```python
# Monta Google Drive en el entorno de Google Colab lo que nos da velocidad a la hora de entrenar al copiar los archivos a Google Collab
from google.colab import drive
drive.mount('/content/drive')

import os
import shutil

# Ruta donde está el dataset en Google Drive
DRIVE_DATA_DIR = "/content/drive/MyDrive/Food-101"

# Ruta donde se copiará el dataset al disco local de Colab.
LOCAL_DATA_DIR = "/content/Food-101"

def dataset_ok(base):
    """
    Verifica que el dataset tenga la estructura esperada:
    - Carpetas 'train' y 'val'
    - Cada una con un número suficiente de clases (~ 101 en Food-101)
    """
    for split in ["train", "val"]:
        # Ruta a la partición (train o val)
        p = os.path.join(base, split)
        # Si no existe la carpeta, el dataset no es válido
        if not os.path.exists(p):
            return False

        # Obtiene todas las carpetas de clases dentro del split
        classes = [
            d for d in os.listdir(p)
            if os.path.isdir(os.path.join(p, d))
        ]

        # Food-101 tiene 101 clases, se permite un pequeño margen
        if len(classes) < 90:
            return False

    return True

print("🔍 Comprobando dataset en Drive...")

# Verifica que el dataset en Google Drive sea válido
if not dataset_ok(DRIVE_DATA_DIR):
    raise RuntimeError("❌ Dataset no válido en Drive")

print("✅ Dataset OK en Drive")

# Si el dataset no existe en el disco local de Colab, se copia
# Esto mejora mucho la velocidad de lectura durante el entrenamiento
if not os.path.exists(LOCAL_DATA_DIR):
    print("📂 Copiando Food-101 a disco local (/content)...")
    shutil.copytree(DRIVE_DATA_DIR, LOCAL_DATA_DIR)
    print("✅ Copia finalizada")
else:
    print("✅ Dataset ya existe en local")

print("🔍 Validando copia local...")

# Se vuelve a verificar el dataset ya copiado
if not dataset_ok(LOCAL_DATA_DIR):
    raise RuntimeError("❌ Dataset inválido en local")

print("✅ Dataset OK en local")

# Variable final que se usará en el entrenamiento
print("📁 DATA_DIR =", LOCAL_DATA_DIR)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
🔍 Comprobando dataset en Drive...
✅ Dataset OK en Drive
📂 Copiando Food-101 a disco local (/content)...
✅ Copia finalizada
🔍 Validando copia local...
✅ Dataset OK en local
📁 DATA_DIR = /content/Food-101
```

Comprobar disponibilidad de GPU

Comandos para comprobar la GPU y la disponibilidad de CUDA desde Python.

- `!nvidia-smi` muestra la GPU del entorno (o no hace nada si no hay GPU).
- `torch.cuda.is_available()` comprueba desde PyTorch.
- Si hay GPU, imprime el nombre del dispositivo.

Nota: importante ejecutar esto antes de iniciar el entrenamiento para confirmar que el entorno está configurado correctamente.

```
# en una celda de notebook
!nvidia-smi || true


import torch
print("cuda_available:", torch.cuda.is_available())
if torch.cuda.is_available():
    print("device:", torch.cuda.get_device_name(0))
```

```
Tue Jan 13 13:01:19 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   40C    P8              9W /   70W |       2MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
cuda_available: True
device: Tesla T4
```

Script de entrenamiento (`train.py`) — descripción y uso

Resumen Script de entrenamiento robusto que soporta varios modos (ej.: `binary`, `food`, `nofood`). Implementa datasets personalizados, aumentos con Albumentations, training loop con AMP (autocast + GradScaler), muestreo balanceado y guardado de checkpoints.

Uso (linea de comandos)

```
python train.py --mode binary --data_dir /path/to/data --no_food_dir /path/to/no_food --model_dir /path/to/models --epochs 12 --t
```

```
#!/usr/bin/env python3
# train.py - Multi-mode: binary | food | nofood
# DEBUG VERSION (robusta y clara)

"""
train.py - Entrenamiento de modelos sobre Food-101

Soporta modos: 'binary' (food vs no-food) y 'food' (clasificación
multi-clase).
Usa timm, albumentations y entrenamiento con AMP.

Ejemplo:
```

```
  python train.py --mode binary --data_dir /content/Food-101 --
no_food_dir /content/drive/MyDrive/no_food --model_dir ./models --
epochs 12
"""


import os, argparse, time
import numpy as np
from PIL import Image, ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
from torch.utils.data import WeightedRandomSampler


from sklearn.model_selection import train_test_split

import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from torch.cuda.amp import GradScaler, autocast

import timm
from torch.optim import AdamW

import albumentations as A
from albumentations.pytorch import ToTensorV2

from tqdm import tqdm
from collections import Counter

# ================================================
# DATASETS
# ================================================
class AlbumentationsImageFolder(Dataset):
    def __init__(self, root_dir, transform, img_size):
        self.root_dir = root_dir
        self.transform = transform
        self.img_size = img_size
        self.samples = []

        classes = sorted(d for d in os.listdir(root_dir)
                         if os.path.isdir(os.path.join(root_dir, d)))

        if len(classes) == 0:
            raise RuntimeError(f"No classes found in {root_dir}")

        self.class_to_idx = {c:i for i,c in enumerate(classes)}
        self.classes = classes

        for c in classes:
```

```
            cdir = os.path.join(root_dir, c)
            for f in os.listdir(cdir):
                self.samples.append((os.path.join(cdir, f),
self.class_to_idx[c]))

        if len(self.samples) == 0:
            raise RuntimeError(f"No images found in {root_dir}")

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        p,y = self.samples[idx]
        try:
            img = np.array(Image.open(p).convert("RGB"))
        except Exception:
            img = np.zeros((self.img_size, self.img_size, 3),
np.uint8)
        img = self.transform(image=img)["image"]
        return img, y


class BinaryFoodNoFoodDataset(Dataset):
    def __init__(self, food_root, transform, img_size,
                 no_food_dir=None, split="train",
                 val_ratio=0.2, seed=42):

        self.transform = transform
        self.img_size = img_size
        self.samples = []

        # FOOD
        for cls in os.listdir(food_root):
            cls_dir = os.path.join(food_root, cls)
            if not os.path.isdir(cls_dir):
                continue
            for f in os.listdir(cls_dir):
                self.samples.append((os.path.join(cls_dir, f), 0))

        # NO FOOD
        if no_food_dir:
            nf_files = []
            for root, _, files in os.walk(no_food_dir):
                for f in files:
                    if f.lower().endswith((".jpg",".png",".jpeg")):
                        nf_files.append(os.path.join(root, f))

            if len(nf_files) > 0:
                tr_nf, va_nf = train_test_split(
```

```
                    nf_files,
                    test_size=val_ratio,
                    random_state=seed,
                    shuffle=True
                )
                use_files = tr_nf if split == "train" else va_nf
                for p in use_files:
                    self.samples.append((p, 1))

        self.classes = ["food", "no_food"]

        if len(self.samples) == 0:
            raise RuntimeError("Binary dataset is empty")

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        p, y = self.samples[idx]
        try:
            img = np.array(Image.open(p).convert("RGB"))
        except Exception:
            img = np.zeros((self.img_size, self.img_size, 3),
np.uint8)
        img = self.transform(image=img)["image"]
        return img, y

# ============================================================
# TRANSFORMS
# ============================================================
def get_transforms(sz):
    size = (sz, sz)

    train_t = A.Compose([
        A.RandomResizedCrop(size=size, scale=(0.7, 1.0)),
        A.HorizontalFlip(p=0.5),
        A.ColorJitter(0.2, 0.2, 0.2, 0.02, p=0.5),
        A.Normalize(mean=(0.485, 0.456, 0.406),
                    std=(0.229, 0.224, 0.225)),
        ToTensorV2()
    ])

    val_t = A.Compose([
        A.Resize(height=sz, width=sz),
        A.Normalize(mean=(0.485, 0.456, 0.406),
                    std=(0.229, 0.224, 0.225)),
        ToTensorV2()
    ])
```

```
    return train_t, val_t

# ======================================================
# TRAIN / EVAL
# ======================================================
def train_epoch(model, loader, opt, scaler, dev, crit, epoch,
log_every):
    model.train()
    total_loss = 0
    seen = 0
    start = time.time()

    print(f"🟢 Epoch {epoch} | {len(loader)} batches")

    for i,(x,y) in enumerate(loader):
        x,y = x.to(dev), y.to(dev)
        opt.zero_grad(set_to_none=True)

        with autocast():
            out = model(x)
            loss = crit(out,y)

        scaler.scale(loss).backward()
        scaler.step(opt)
        scaler.update()

        total_loss += loss.item() * x.size(0)
        seen += x.size(0)

        if i % log_every == 0:
            print(f"   batch {i}/{len(loader)} | loss
{loss.item():.4f}")

    avg = total_loss / seen
    print(f"🟢 Epoch {epoch} done | avg train loss {avg:.4f} |
{time.time()-start:.1f}s")
    return avg

def eval_model(model, loader, dev):
    model.eval()
    ok = tot = 0
    with torch.no_grad():
        for x,y in loader:
            x,y = x.to(dev), y.to(dev)
            p = model(x).argmax(1)
            ok += (p==y).sum().item()
            tot += y.size(0)
    return ok / tot
```

```
# ====================================================
# MAIN
# ====================================================
# ------------------ PEGAR/REEMPLAZAR main() POR ESTE BLOQUE --------
----------
import shutil
import sys

def parse_args_robust():
    ap = argparse.ArgumentParser()
    ap.add_argument("--mode", required=True)
    ap.add_argument("--data_dir")
    ap.add_argument("--no_food_dir")
    ap.add_argument("--model_dir", default="./models")
    ap.add_argument("--epochs", type=int, default=12)
    ap.add_argument("--bs", type=int, default=16)
    ap.add_argument("--img_size", type=int, default=192)
    ap.add_argument("--workers", type=int, default=2)
    ap.add_argument("--log_every", type=int, default=200)
    ap.add_argument("--copy_to_tmp", action="store_true",
                    help="Copiar dataset a /tmp para evitar lentitud
de Google Drive (recomendado).")
    ap.add_argument("--save_every_epoch", type=int, default=1,
                    help="Guardar checkpoint cada N épocas.")
    # Si ejecutas desde bash con args funcionará como antes.
    # Si ejecutas dentro de IPython/Colab y no pasas args, evitamos
SystemExit y devolvemos None.
    try:
        args = ap.parse_args()
    except SystemExit:
        # Estamos en entorno interactivo sin args: devolver None para
que el entorno (shell) decida.
        print(" argparse: falta --mode o estás en entorno
interactivo. Usa la ejecución por línea de comandos (%%bash) o pasa
args explícitos.", flush=True)
        raise
    return args

def main():
    args = parse_args_robust()

    # Si pides copiar a tmp (recomendado en Colab cuando tu dataset
está en Drive)
    if args.copy_to_tmp and args.data_dir:
        tmp_dest = f"/tmp/food_dataset_{int(time.time())}"
        if os.path.exists(tmp_dest):
            print("🗑 Limpiando tmp anterior...", flush=True)
            shutil.rmtree(tmp_dest)
```

```
        print(f"Copiando datos desde {args.data_dir} -> {tmp_dest}
(esto puede tardar unos minutos)...", flush=True)
        shutil.copytree(args.data_dir, tmp_dest)
        args.data_dir = tmp_dest
        print("Copia completada.", flush=True)

    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    print(f"DEVICE: {device}", flush=True)

    tr_t, va_t = get_transforms(args.img_size)

    if args.mode == "food":
        tr_ds =
AlbumentationsImageFolder(os.path.join(args.data_dir,"train"), tr_t,
args.img_size)
        va_ds =
AlbumentationsImageFolder(os.path.join(args.data_dir,"val"), va_t,
args.img_size)
        classes = tr_ds.classes
        with open(os.path.join(args.model_dir,
f"classes_{args.mode}.txt"), "w") as f:
            f.write("\n".join(classes))
    elif args.mode == "binary":
        tr_ds =
BinaryFoodNoFoodDataset(os.path.join(args.data_dir,"train"), tr_t,
args.img_size, args.no_food_dir,"train")
        va_ds =
BinaryFoodNoFoodDataset(os.path.join(args.data_dir,"val"), va_t,
args.img_size, args.no_food_dir,"val")
        classes = ["food","no_food"]
    else:
        raise ValueError("Modo no soportado")

    print(f"Train samples: {len(tr_ds)}", flush=True)
    print(f"Val samples:   {len(va_ds)}", flush=True)
    print(f"Classes: {len(classes)}", flush=True)

    counts = Counter(y for _,y in tr_ds.samples)
    print("Class distribution:", counts, flush=True)

    sample_weights = [1.0 / counts[y] for _, y in tr_ds.samples]
    sampler = WeightedRandomSampler(
        weights=sample_weights,
        num_samples=len(sample_weights),
        replacement=True
    )
```

```
    # Si estás leyendo desde Drive, para debug pon workers=0 (evita
deadlocks)
    workers = args.workers
    if 'google.colab' in sys.modules:
        print("Detectado Colab: si tienes problemas con DataLoader
prueba --workers 0", flush=True)

    tr_dl = DataLoader(
        tr_ds,
        batch_size=args.bs,
        sampler=sampler,
        num_workers=workers,
        pin_memory=(device.type == "cuda"),
        persistent_workers=False
    )
    va_dl = DataLoader(va_ds, args.bs, shuffle=False,
                       num_workers=workers, pin_memory=(device.type
== "cuda"))

    model = timm.create_model("efficientnet_b0", pretrained=True,
                              num_classes=len(classes)).to(device)

    opt = AdamW(model.parameters(), lr=3e-4)
    scaler = GradScaler()
    crit = nn.CrossEntropyLoss(label_smoothing=0.1)

    best = 0
    os.makedirs(args.model_dir, exist_ok=True)

    try:
        for e in range(1, args.epochs+1):
            # log inicial por epoch
            print(f"\n===  ◇  Época {e}/{args.epochs} ===",
flush=True)
            print(f"🟡 Epoch {e} | {len(tr_dl)} batches
(bs={args.bs})", flush=True)
            train_epoch(model, tr_dl, opt, scaler, device, crit, e,
args.log_every)
            acc = eval_model(model, va_dl, device)
            print(f"📈 VAL ACC = {acc:.4f}", flush=True)

            # guardado por epochs según parámetro
            if e % args.save_every_epoch == 0:
                ckpt = {
                    "epoch": e,
                    "state_dict": model.state_dict(),
                    "optimizer": opt.state_dict(),
                    "scaler": scaler.state_dict(),
                    "acc": acc
```

```
                }
                path = os.path.join(args.model_dir,
f"checkpoint_epoch{e}_{args.mode}.pth")
                torch.save(ckpt, path)
                print(f"💾 Checkpoint guardado: {path}", flush=True)

            if acc > best:
                best = acc
                torch.save(model.state_dict(),
                           os.path.join(args.model_dir,
f"best_{args.mode}.pth"))
                print("NEW BEST MODEL SAVED", flush=True)

    except Exception as exc:
        # guardar checkpoint parcial y reportar excepción
        print("Excepción durante el entrenamiento, guardando
checkpoint parcial...", flush=True)
        try:
            torch.save({
                "state_dict": model.state_dict(),
                "optimizer": opt.state_dict(),
                "scaler": scaler.state_dict(),
                "exception": str(exc)
            }, os.path.join(args.model_dir,
f"crash_partial_{args.mode}.pth"))
            print("Checkpoint parcial guardado.", flush=True)
        except Exception as e2:
            print("No se pudo guardar checkpoint parcial:", e2,
flush=True)
        raise

    print("-> Finished | Best acc:", best, flush=True)

# ------------------------------------------------------------------
--------
```

EJECUCIÓN DEL TRAIN EN 3 CAPAS

Celda 5 — Ejecutar entrenamiento (modo `binary`)

```
## Ejecutar entrenamiento — modo `binary` (ejemplo)

Este bloque crea variables de entorno y ejecuta `train.py` para el modo `binary`.

- `LOCAL_DATA_PATH`: dataset copiado localmente (`/content/Food-101`).
- `NO_FOOD_DIR`: carpeta con imágenes no-food en Drive (sin split).
- `MODEL_DIR`: carpeta de salida en Drive (para guardar checkpoints y logs).
```

Consejo

- Ajusta `--bs`, `--img_size`, `--epochs` según la GPU.
- Usa `--copy_to_tmp` si tu dataset está aún en Drive y quieres copiarlo a `/tmp` para mejor I/O.

Entrenamiento modo `food` (fases)

Comando que entrena la clasificación multi-clase (Food-101) posiblemente en dos fases:

- `--phase both` y parámetros `--epochs_phase1`, `--epochs_phase2` (si tu script lo soporta).

Notas

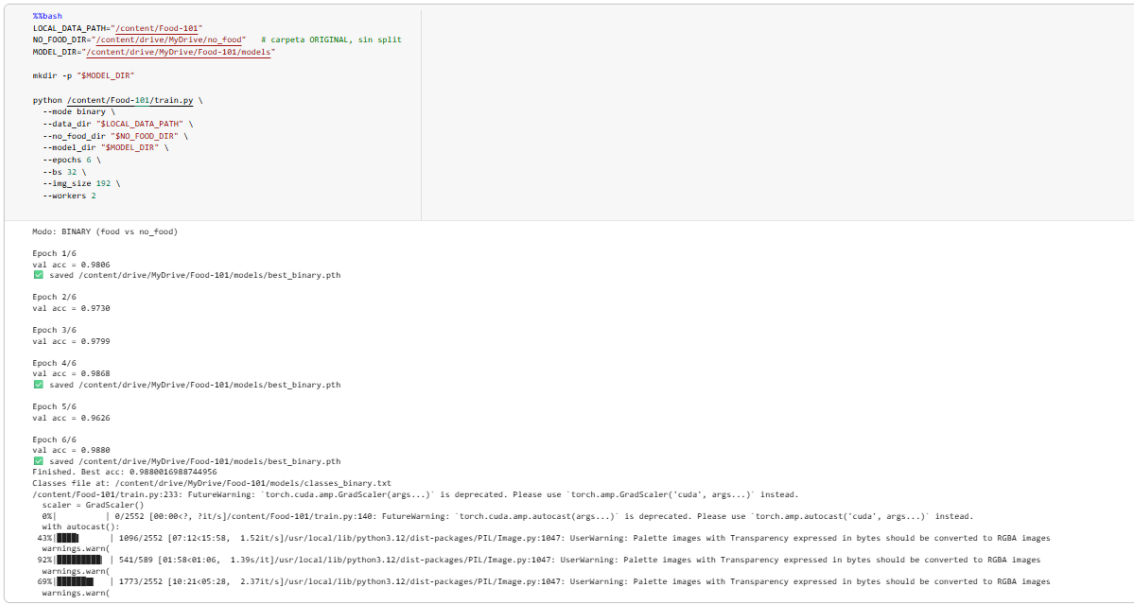
- Comprueba que `train.py` implemente la lógica de fases (`phase`, `epochs_phase1`, `epochs_phase2`).
- Guardar checkpoints en Drive permite reanudar o descargar los mejores modelos.

Entrenamiento modo `nofood`

Similar a los anteriores: ejecuta `train.py` en un modo orientado a entrenar un clasificador que trate sólo clases de "no food" o un modelo especializado.

Atención

- Asegúrate de que `--mode nofood` esté implementado en `train.py`.



```
%%bash
LOCAL_DATA_PATH="/content/Food-101"
NO_FOOD_DIR="/content/drive/MyDrive/no_food"   # carpeta ORIGINAL, sin split
MODEL_DIR="/content/drive/MyDrive/Food-101/models"

mkdir -p "$MODEL_DIR"

python /content/Food-101/train.py \
  --mode binary \
  --data_dir "$LOCAL_DATA_PATH" \
  --no_food_dir "$NO_FOOD_DIR" \
  --model_dir "$MODEL_DIR" \
  --epochs 6 \
  --bs 32 \
  --img_size 192 \
  --workers 2
```

```
Modo: BINARY (food vs no_food)

Epoch 1/6
val acc = 0.9806
✅ saved /content/drive/MyDrive/Food-101/models/best_binary.pth

Epoch 2/6
val acc = 0.9730

Epoch 3/6
val acc = 0.9799

Epoch 4/6
val acc = 0.9868
✅ saved /content/drive/MyDrive/Food-101/models/best_binary.pth

Epoch 5/6
val acc = 0.9626

Epoch 6/6
val acc = 0.9880
✅ saved /content/drive/MyDrive/Food-101/models/best_binary.pth
Finished. Best acc: 0.9880016988744956
Classes file at: /content/drive/MyDrive/Food-101/models/classes_binary.txt
/content/Food-101/train.py:233: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradScaler('cuda', args...)` instead.
  scaler = GradScaler()
  0%|          | 0/2552 [00:00<?, ?it/s]/content/Food-101/train.py:140: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with autocast():
 43%|████▎     | 1096/2552 [07:12<15:58,  1.52it/s]/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
 92%|█████████▏| 541/589 [01:58<01:06,  1.39s/it]/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
 69%|██████▊   | 1773/2552 [10:21<05:28,  2.37it/s]/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
```

```
%%bash
python train.py \
  --mode food \
  --data_dir "/content/Food-101" \
  --model_dir "/content/drive/MyDrive/Food-101/models" \
  --phase both \
  --epochs_phase1 10 \
  --epochs_phase2 20 \
  --bs 16 \
  --img_size 192 \
  --save_ckpt_to_drive
```

```
    spaghetti_bolognese     0.0000    0.0000    0.0000         0
    spaghetti_carbonara     0.0000    0.0000    0.0000         0
          spring_rolls      0.0000    0.0000    0.0000         0
                 steak      0.0000    0.0000    0.0000         0
  strawberry_shortcake      0.0000    0.0000    0.0000         0
                 sushi      0.0000    0.0000    0.0000         0
                 tacos      0.0000    0.0000    0.0000         0
              takoyaki      0.0000    0.0000    0.0000         0
              tiramisu      0.0000    0.0000    0.0000         0
    tortilla_de_patata      0.0000    0.0000    0.0000         0
          tuna_tartare      0.0000    0.0000    0.0000         0
               waffles      0.0000    0.0000    0.0000         0

              accuracy                           0.0024     15147
             macro avg      0.0025    0.0020    0.0019     15147
          weighted avg      0.0030    0.0024    0.0023     15147

💾 Checkpoint guardado: /content/drive/MyDrive/Food-101/models/checkpoint_phase1_epoch6_food.pth

=== 🔸 phase1 Epoch 7/10 ===
🟡 Epoch 7 | 4560 batches
   batch 0/4560 | loss 1.1350
   batch 200/4560 | loss 1.1974
   batch 400/4560 | loss 1.1222
   batch 600/4560 | loss 1.3765
   batch 800/4560 | loss 1.0213
   batch 1000/4560 | loss 1.2715
   batch 1200/4560 | loss 1.2554
   batch 1400/4560 | loss 1.3456
   batch 1600/4560 | loss 1.1307
   batch 1800/4560 | loss 1.2718
   batch 2000/4560 | loss 1.3800
   batch 2200/4560 | loss 1.3484
   batch 2400/4560 | loss 1.0267
   batch 2600/4560 | loss 1.3855
   batch 2800/4560 | loss 1.2845
   batch 3000/4560 | loss 1.0363
   batch 3200/4560 | loss 1.5966
   batch 3400/4560 | loss 1.1607
   batch 3600/4560 | loss 1.5027
   batch 3800/4560 | loss 1.6243
   batch 4000/4560 | loss 1.2799
   batch 4200/4560 | loss 1.4208
   batch 4400/4560 | loss 1.3827
🟢 Epoch 7 done | avg train loss 1.2689 | 587.9s
📈 VAL ACC = 0.0015
--- Per-class report:
                          precision    recall  f1-score   support
```

```
%%bash
NO_FOOD_DIR="/content/drive/MyDrive/no_food"
MODEL_DIR="/content/drive/MyDrive/Food-101/models"

python /content/drive/MyDrive/Food-101/train.py \
  --mode nofood \
  --no_food_dir "$NO_FOOD_DIR" \
  --model_dir "$MODEL_DIR" \
  --epochs 12 \
  --bs 16 \
  --img_size 192 \
  --workers 2
```

```
Modo: NOFOOD (tipos dentro de no-food)

Epoch 1/12
val acc = 0.8189
✅ saved /content/drive/MyDrive/Food-101/models/best_nofood.pth

Epoch 2/12
val acc = 0.8825
✅ saved /content/drive/MyDrive/Food-101/models/best_nofood.pth

Epoch 3/12
val acc = 0.9110
✅ saved /content/drive/MyDrive/Food-101/models/best_nofood.pth

Epoch 4/12
val acc = 0.9346
✅ saved /content/drive/MyDrive/Food-101/models/best_nofood.pth

Epoch 5/12
val acc = 0.9419
✅ saved /content/drive/MyDrive/Food-101/models/best_nofood.pth

Epoch 6/12
val acc = 0.9477
✅ saved /content/drive/MyDrive/Food-101/models/best_nofood.pth

Epoch 7/12
val acc = 0.9427

Epoch 8/12
val acc = 0.9491
✅ saved /content/drive/MyDrive/Food-101/models/best_nofood.pth

Epoch 9/12
val acc = 0.9601
✅ saved /content/drive/MyDrive/Food-101/models/best_nofood.pth

Epoch 10/12
val acc = 0.9560

Epoch 11/12
val acc = 0.9598

Epoch 12/12
val acc = 0.9566
Finished. Best acc: 0.9600734725444702
Classes file at: /content/drive/MyDrive/Food-101/models/classes_nofood.txt
/content/drive/MyDrive/Food-101/train.py:248: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradScaler('cuda', args...)` instead.
  scaler = GradScaler()
  0%|          | 0/647 [00:00<?, ?it/s]/content/drive/MyDrive/Food-101/train.py:140: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with autocast():
 58%|█████▊    | 374/647 [45:01<31:24,  6.90s/it]/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
 78%|███████▊  | 503/647 [1:00:19<16:08,  6.73s/it]/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
 19%|█▊        | 120/647 [01:47<09:01,  1.03s/it]/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
 27%|██▋       | 175/647 [02:40<07:41,  1.02it/s]/usr/local/lib/python3.12/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
```

VALIDACIÓN DEL TRAIN

Inferencia rápida con el clasificador binario

Script de ejemplo que:

1. Carga `CLASSES_BIN` desde `classes_binary.txt`.
2. Crea el modelo `efficientnet_b0` con `timm` y carga pesos (`best_binary.pth`).
3. Define transform y función `predict(path)` que devuelve la etiqueta, probabilidad y vector de probabilidades.

Uso

- Actualiza `MODEL_DIR` y nombres de checkpoint (`BIN_CKPT`) según tu árbol de ficheros.
- Prueba con algunos ejemplos (3 food / 3 no-food) para verificar resultados.

Salida

- Para cada imagen imprime: `filename -> predicted_class probability [all_probs...]`.

```
import torch, timm, os
from PIL import Image
from torchvision import transforms
import numpy as np

MODEL_DIR = "/content/drive/MyDrive/Food-101/models"
BIN_CKPT = os.path.join(MODEL_DIR, "best_binary.pth")
CLASSES_BIN = os.path.join(MODEL_DIR, "classes_binary.txt")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("device:", device)

# Cargar clases
with open(CLASSES_BIN, "r") as f:
    classes_bin = [l.strip() for l in f if l.strip()]
print("Loaded binary classes:", classes_bin)

# crear modelo y cargar pesos
m = timm.create_model("efficientnet_b0", pretrained=False, num_classes=len(classes_bin))
sd = torch.load(BIN_CKPT, map_location="cpu")
# quitar posible prefijo module.
new = { (k[len("module."): ] if k.startswith("module.") else k): v for k,v in sd.items() }
m.load_state_dict(new, strict=False)
m.to(device).eval()

# transform
IMG_SIZE = 192
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.485,0.456,0.406), std=(0.229,0.224,0.225))
])

# toma unas imágenes de prueba (3 de food val y 3 de no-food)
food_examples = []
val_food_dir = "/content/Food-101/val"
for root, dirs, files in os.walk(val_food_dir):
    if files:
        for fn in files[:3]:
            food_examples.append(os.path.join(root, fn))
    if len(food_examples) >= 3:
        break

nofood_examples = []
no_food_dir = "/content/drive/MyDrive/no_food"
# toma 3 imágenes de distintas subcarpetas
cnt = 0
for sub in sorted(os.listdir(no_food_dir)):
    sp = os.path.join(no_food_dir, sub)
    if os.path.isdir(sp):
        fs = [f for f in os.listdir(sp) if f.lower().endswith((".jpg",".png",".jpeg"))]
        if fs:
            nofood_examples.append(os.path.join(sp, fs[0]))
            cnt += 1
    if cnt >= 3:
        break

print("Food examples:", food_examples)
print("No-food examples:", nofood_examples)
```

```
def predict(path):
    img = Image.open(path).convert("RGB")
    x = transform(img).unsqueeze(0).to(device)
    with torch.no_grad():
        out = m(x)
        probs = torch.softmax(out, dim=1).cpu().numpy()[0]
        p = int(out.argmax(1).cpu().numpy()[0])
    return classes_bin[p], float(probs[p]), probs

print("\n--- PREDICTIONS FOOD EXAMPLES ---")
for p in food_examples:
    cls, prob, probs = predict(p)
    print(os.path.basename(p), "->", cls, prob, "all_probs:", probs.tolist())

print("\n--- PREDICTIONS NO-FOOD EXAMPLES ---")
for p in nofood_examples:
    cls, prob, probs = predict(p)
    print(os.path.basename(p), "->", cls, prob, "all_probs:", probs.tolist())
```

```
device: cuda
Loaded binary classes: ['food', 'no_food']
Food examples: []
No-food examples: ['/content/drive/MyDrive/no_food/animals/animals_373_1a3b8cb8.jpeg', '/content/drive/MyDrive/no_food/beaches/beaches_397_43fd4aa9.jpg', '/content/drive/MyDrive/no_food/bottles/bottles_286_6d2ac56a.jpg']

--- PREDICTIONS FOOD EXAMPLES ---

--- PREDICTIONS NO-FOOD EXAMPLES ---
animals_373_1a3b8cb8.jpeg -> no_food 0.9988183379173279 all_probs: [0.0011816353071480099, 0.9988183379173279]
beaches_397_43fd4aa9.jpg -> no_food 0.9825178980827332 all_probs: [0.017482107505202293, 0.9825178980827332]
bottles_286_6d2ac56a.jpg -> no_food 0.9988580942153931 all_probs: [0.0011419359361752868, 0.9988580942153931]
```

Comprobar GPU, logs y ficheros de models

Comandos útiles:

- `nvidia-smi` — comprobar GPU.
- `tail -n 80 /content/drive/.../train.log` — ver últimas líneas del log de entrenamiento.
- `ls -la /content/drive/.../models` — ver checkpoints guardados.

Incluye en la entrega una captura o listado con los checkpoints y el log más relevante.

```
%%bash
# comprueba GPU
nvidia-smi || true

# ver últimas líneas del log
tail -n 80 /content/drive/MyDrive/Food-101/models/train.log || true

# ver checkpoint guardado
ls -la /content/drive/MyDrive/Food-101/models || true
```

```
Mon Jan 12 12:25:40 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   76C    P0             31W /   70W |     188MiB /  15360MiB |      4%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
+-----------------------------------------------------------------------------------------+
Training started: Mon Jan  5 13:37:17 2026
Config: IMG_SIZE=224, BS=32, EPOCHS=12
Training started: Mon Jan  5 14:40:20 2026
Config: IMG_SIZE=224, BS=32, EPOCHS=12
Epoch 1/12 — train_loss: 2.8880 val_top1: 0.5495 val_top3: 0.7211 time: 15368.5s
Epoch 2/12 — train_loss: 1.9605 val_top1: 0.7254 val_top3: 0.8656 time: 728.4s
Epoch 3/12 — train_loss: 1.6667 val_top1: 0.7577 val_top3: 0.8876 time: 708.5s
Epoch 4/12 — train_loss: 1.5200 val_top1: 0.7740 val_top3: 0.8973 time: 707.7s
Training started: Tue Jan  6 10:39:10 2026
Config: IMG_SIZE=192, BS=16, EPOCHS=6
Epoch 5/6 — train_loss: 1.6230 val_top1: 0.7473 val_top3: 0.8769 time: 462.6s
Epoch 6/6 — train_loss: 1.5540 val_top1: 0.7517 val_top3: 0.8815 time: 431.5s
Training started: Thu Jan  8 09:40:34 2026
Config: IMG_SIZE=192, BS=16, EPOCHS=20
total 180602
-rw------- 1 root root 16335847 Jan 11 16:53 best_binary.pth
-rw------- 1 root root 16956871 Jan 11 11:44 best_efficientnet_b0_122cls.pth
-rw------- 1 root root 16948869 Jan 10 17:18 best_efficientnet_b0_backup.pth
-rw------- 1 root root 16948869 Jan 10 08:45 best_efficientnet_b0.pth
-rw------- 1 root root 16950603 Jan 12 05:27 best_food.pth
-rw------- 1 root root 16950329 Jan 10 16:17 best_model.pth
-rw------- 1 root root 16459431 Jan 12 11:35 best_nofood.pth
-rw------- 1 root root 16930725 Jan 10 14:43 best.pth
-rw------- 1 root root 50442424 Jan 10 14:52 checkpoint.pth
-rw------- 1 root root       12 Jan 11 15:02 classes_binary.txt
-rw------- 1 root root     1419 Jan 11 23:13 classes_food.txt
-rw------- 1 root root      242 Jan 12 07:53 classes_nofood.txt
-rw------- 1 root root     1420 Jan 10 11:09 classes.txt
drwx------ 2 root root     4096 Jan  6 16:45 .ipynb_checkpoints
-rw------- 1 root root      445 Jan  6 10:54 metrics.csv
-rw------- 1 root root      813 Jan  8 09:50 train.log
```

Verificación: existencia de carpetas importantes

La celda imprime si `/content/Food-101` y la subcarpeta `val` existen, y muestra los primeros ficheros de `val`.

Por qué incluirlo

- Esto demuestra en el notebook que el dataset está correctamente ubicado antes de entrenar.

```
import os
print("VERIFICACIÓN:")
print("/content/Food-101/ existe:", os.path.exists("/content/Food-101"))
print("/content/Food-101/val existe:", os.path.exists("/content/Food-101/val"))
print("Contenido val:", os.listdir("/content/Food-101/val")[:5] if os.path.exists("/content/Food-101/val") else "NO")
```

Exploración rápida de la estructura del dataset

Comandos:

- `!ls -la /content/Food-101`
- `!find /content/Food-101 -maxdepth 3 -type d | head -n 50`
- Código Python para localizar `train`, `val`, `test`.

Objetivo

- Mostrar que la estructura es `root/train/<class>/*`, `root/val/<class>/*` (requisito del dataset Food-101 para este script).

```
!ls -la /content/Food-101
!find /content/Food-101 -maxdepth 3 -type d | head -n 50

import os

for root, dirs, files in os.walk("/content/Food-101"):
    if 'val' in dirs:
        print("✅ VAL encontrada en:", os.path.join(root, 'val'))
    if 'train' in dirs:
        print("✅ TRAIN encontrada en:", os.path.join(root, 'train'))
    if 'test' in dirs:
        print("✅ TEST encontrada en:", os.path.join(root, 'test'))
```

Limpiar checkpoint y guardar modelo sin classifier antiguo

Explicación:

- Se cargan `classes.txt` y el checkpoint (`best_efficientnet_b0.pth`).
- Se crea la arquitectura `efficientnet_b0` con `num_classes` = len(classes).
- Se limpia el `state_dict` del checkpoint eliminando claves de `classifier.` y prefijos `module.`.
- Esto permite reutilizar pesos de la backbone aunque el clasificador del checkpoint antiguo tenga otra forma.
- Finalmente se guarda un nuevo estado con `model.state_dict()`.

Motivo

- Útil cuando el checkpoint proviene de un entrenamiento con un `classifier` distinto (por ejemplo, 1000 clases -> 122 clases).

```python
import torch, timm, os
from collections import OrderedDict

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

CLASSES_PATH = "/content/drive/MyDrive/Food-101/models/classes.txt"
MODEL_PATH   = "/content/drive/MyDrive/Food-101/models/best_efficientnet_b0.pth"

# 1 Clases
with open(CLASSES_PATH) as f:
    classes = [l.strip() for l in f if l.strip()]
num_classes = len(classes)
print("Loaded classes:", num_classes)

# 2 Modelo
model = timm.create_model(
    "efficientnet_b0",
    pretrained=False,
    num_classes=num_classes
).to(DEVICE)

# 3 Cargar checkpoint
ckpt = torch.load(MODEL_PATH, map_location="cpu")
state_dict = ckpt if isinstance(ckpt, dict) else ckpt.state_dict()

# 4 LIMPIAR classifier del checkpoint (CLAVE)
clean = OrderedDict()
for k, v in state_dict.items():
    k = k.replace("module.", "")
    if k.startswith("classifier."):
        continue   # ⬅️ ELIMINAMOS classifier viejo
    clean[k] = v

# 5 Cargar pesos
missing, unexpected = model.load_state_dict(clean, strict=False)
model.eval()

print("✅ Modelo cargado SIN classifier antiguo")
print("Missing keys:", missing)
print("Unexpected keys:", unexpected)

# 6 Guardar modelo listo
OUT_PATH = "/content/drive/MyDrive/Food-101/models/best_efficientnet_b0_122cls.pth"
torch.save(model.state_dict(), OUT_PATH)
print("💾 Guardado en:", OUT_PATH)
```

```
Loaded classes: 122
✅ Modelo cargado SIN classifier antiguo
Missing keys: ['classifier.weight', 'classifier.bias']
Unexpected keys: []
💾 Guardado en: /content/drive/MyDrive/Food-101/models/best_efficientnet_b0_122cls.pth
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Inference en cascada (binary → food / nofood)

Descripción:

- `inference_cascade.py` implementa una tubería de inferencia en 2 pasos:
  1. El modelo binario (`best_binary.pth`) decide si la imagen es `food` o `no_food`.
  2. Si es `food`, se consulta `food_model` para clasificar la clase concreta. Si no es `food`, se consulta `no_model` para predecir la clase no-food más probable.

Funciones clave

- `load_model(ckpt, num_classes)` — carga un modelo desde checkpoint y limpia prefijo `module.`.
- `predict_single(path, bin_thresh=None)`:
  - Si `bin_thresh` es `None` usa argmax binario.
  - Si `bin_thresh` es float interpreta el umbral de probabilidad para la clase `food`.

Salida

- Tupla con (`"food"|"no_food"`, etiqueta_predicha, probabilidad, lista_de_probabilidades).

Ejemplo

```
print(predict_single("/path/to/image.jpg", bin_thresh=0.5))
```

```python
# inference_cascade.py
import os, torch, timm, numpy as np
from PIL import Image
from torchvision import transforms

MODEL_DIR = "/content/drive/MyDrive/Food-101/models"
BIN_CKPT = os.path.join(MODEL_DIR, "best_binary.pth")
FOOD_CKPT = os.path.join(MODEL_DIR, "best_food.pth")
NOFOOD_CKPT = os.path.join(MODEL_DIR, "best_nofood.pth")

CLASSES_BIN = [l.strip() for l in open(os.path.join(MODEL_DIR,"classes_binary.txt")) if l.strip()]
CLASSES_FOOD = [l.strip() for l in open(os.path.join(MODEL_DIR,"classes_food.txt")) if l.strip()]
CLASSES_NO = [l.strip() for l in open(os.path.join(MODEL_DIR,"classes_nofood.txt")) if l.strip()]

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def load_model(ckpt, num_classes):
    m = timm.create_model("efficientnet_b0", pretrained=False, num_classes=num_classes)
    sd = torch.load(ckpt, map_location="cpu")
    sd = { (k[len("module."): ] if k.startswith("module.") else k): v for k,v in sd.items() }
    m.load_state_dict(sd, strict=False)
    return m.to(device).eval()

bin_model = load_model(BIN_CKPT, len(CLASSES_BIN))
food_model = load_model(FOOD_CKPT, len(CLASSES_FOOD))
no_model = load_model(NOFOOD_CKPT, len(CLASSES_NO))

IMG_SIZE = 192
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE,IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.485,0.456,0.406), std=(0.229,0.224,0.225))
])

def predict_single(path, bin_thresh=None):
    img = Image.open(path).convert("RGB")
    x = transform(img).unsqueeze(0).to(device)
    with torch.no_grad():
        out_bin = torch.softmax(bin_model(x), dim=1).cpu().numpy()[0]
    # bin prediction
    idx_bin = int(out_bin.argmax())
    if bin_thresh is not None:
        # interpret bin_thresh as probability threshold for "food"
        p_food = out_bin[ CLASSES_BIN.index("food") ]
        is_food = p_food >= bin_thresh
    else:
        is_food = (idx_bin == CLASSES_BIN.index("food"))
    if is_food:
        with torch.no_grad():
            out_food = torch.softmax(food_model(x), dim=1).cpu().numpy()[0]
        pred_idx = int(out_food.argmax())
        return ("food", CLASSES_FOOD[pred_idx], float(out_food[pred_idx]), out_food.tolist())
    else:
        with torch.no_grad():
            out_no = torch.softmax(no_model(x), dim=1).cpu().numpy()[0]
        pred_idx = int(out_no.argmax())
        return ("no_food", CLASSES_NO[pred_idx], float(out_no[pred_idx]), out_no.tolist())

# ejemplo
# print(predict_single("/content/drive/MyDrive/some_image.jpg", bin_thresh=0.5))
```

EVALUACIÓN FINAL

Evaluación final del modelo

Propósito: **Evaluar** el modelo de clasificación de alimentos entrenado sobre Food-101 usando **exclusivamente las clases reales** del entrenamiento.

Qué hace:

- Carga **configuración**, **clases** y** modelo entrenado**.
- **Garantiza** que el **número** y **orden** de clases coinciden con el entrenamiento.
- Realiza **inferencia **sobre todo el **conjunto de test**.
- Calcula **métricas Top-1, Top-3 **y **Top-5** Accuracy.
- **Genera** un **classification report detallado**.

```
# ============================================================
# EVALUACIÓN FINAL — Food-101 (CLASES REALES DE ENTRENAMIENTO)
# ============================================================
import os
import torch
import time
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from torchvision import transforms
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    classification_report,
    top_k_accuracy_score
)
import seaborn as sns

# ------------------------------------------------------------
# CONFIGURACION
# ------------------------------------------------------------
TRAIN_DIR = "/content/drive/MyDrive/Food-101/train"
TEST_DIR  = "/content/drive/MyDrive/Food-101/test"
MODEL_PATH = "/content/drive/MyDrive/Food-101/models/best_food.pth"
CLASSES_FILE = "/content/drive/MyDrive/Food-101/models/classes_food.txt"

IMG_SIZE = 192
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("- Device:", DEVICE)

# ------------------------------------------------------------
# CLASES (FUENTE UNICA: ENTRENAMIENTO REAL)
# ------------------------------------------------------------
if not os.path.exists(CLASSES_FILE):
    raise FileNotFoundError(f"No existe {CLASSES_FILE}")

with open(CLASSES_FILE, "r") as f:
    classes = [l.strip() for l in f if l.strip()]

NUM_CLASSES = len(classes)
class_to_idx = {c: i for i, c in enumerate(classes)}

print("- Clases reales del modelo:", NUM_CLASSES)

# ------------------------------------------------------------
# MODELO
# ------------------------------------------------------------
ckpt = torch.load(MODEL_PATH, map_location="cpu")

# extraer state_dict correctamente
if isinstance(ckpt, dict) and "state_dict" in ckpt:
    state_dict = ckpt["state_dict"]
else:
    state_dict = ckpt

# limpiar posible prefijo "module."
state_dict = {k.replace("module.", ""): v for k, v in state_dict.items()}

# crear modelo con el MISMO nº de clases
model = timm.create_model(
    "efficientnet_b0",
    num_classes=NUM_CLASSES
)

# cargar pesos COMPLETOS
model.load_state_dict(state_dict, strict=True)
model.to(DEVICE).eval()

print("- Modelo cargado correctamente")

# ------------------------------------------------------------
# TRANSFORMS (idénticos a validación)
# ------------------------------------------------------------
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=(0.485, 0.456, 0.406),
        std=(0.229, 0.224, 0.225)
    )
])

# ------------------------------------------------------------
# PREDICCION
# ------------------------------------------------------------
def predict(img_path):
    img = Image.open(img_path).convert("RGB")
    x = transform(img).unsqueeze(0).to(DEVICE)
    with torch.no_grad():
        out = model(x)
        probs = torch.softmax(out, dim=1)[0].cpu().numpy()
    return probs

    num_classes=NUM_CLASSES
)

# cargar pesos COMPLETOS
model.load_state_dict(state_dict, strict=True)
model.to(DEVICE).eval()

print("- Modelo cargado correctamente")

# ------------------------------------------------------------
# TRANSFORMS (idénticos a validación)
# ------------------------------------------------------------
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=(0.485, 0.456, 0.406),
        std=(0.229, 0.224, 0.225)
    )
])
```

```
# ------------------------------------------------------------
# PREDICCION
# ------------------------------------------------------------
def predict(img_path):
    img = Image.open(img_path).convert("RGB")
    x = transform(img).unsqueeze(0).to(DEVICE)
    with torch.no_grad():
        out = model(x)
        probs = torch.softmax(out, dim=1)[0].cpu().numpy()
    return probs

# ------------------------------------------------------------
# EVALUACIÓN COMPLETA
# ------------------------------------------------------------
y_true, y_pred, y_probs = [], [], []

for cls in os.listdir(TEST_DIR):
    if cls not in class_to_idx:
        continue  # ← ignora clases no entrenadas

    gt = class_to_idx[cls]
    cls_path = os.path.join(TEST_DIR, cls)

    for fn in os.listdir(cls_path):
        if not fn.lower().endswith((".jpg", ".jpeg", ".png")):
            continue

        path = os.path.join(cls_path, fn)
        try:
            probs = predict(path)
            y_true.append(gt)
            y_pred.append(int(np.argmax(probs)))
            y_probs.append(probs)
        except Exception as e:
            print(" Error:", path, e)

y_true = np.array(y_true)
y_pred = np.array(y_pred)
y_probs = np.array(y_probs)

print(" Total samples evaluados:", len(y_true))

# ------------------------------------------------------------
# METRICAS
# ------------------------------------------------------------
labels = list(range(NUM_CLASSES))

top1 = accuracy_score(y_true, y_pred)
top3 = top_k_accuracy_score(y_true, y_probs, k=3, labels=labels)
top5 = top_k_accuracy_score(y_true, y_probs, k=5, labels=labels)

print(f"1. Top-1 Accuracy : {top1:.4f}")
print(f"2. Top-3 Accuracy : {top3:.4f}")
print(f"3. Top-5 Accuracy : {top5:.4f}")

print("\n Classification Report:")
print(classification_report(
    y_true,
    y_pred,
    target_names=classes,
    zero_division=0
))

# ------------------------------------------------------------
# CONFUSION MATRIX (TOP 15 CLASES MAS FRECUENTES)
# ------------------------------------------------------------
cm = confusion_matrix(y_true, y_pred)
idx = np.argsort(cm.sum(axis=1))[-15:]

plt.figure(figsize=(12, 10))
sns.heatmap(
    cm[np.ix_(idx, idx)],
    xticklabels=[classes[i] for i in idx],
    yticklabels=[classes[i] for i in idx],
    cmap="Blues",
    annot=False
)
plt.title("Confusion Matrix — Top 15 clases")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

```
- Device: cuda
- Clases reales del modelo: 122
- Modelo cargado correctamente
```

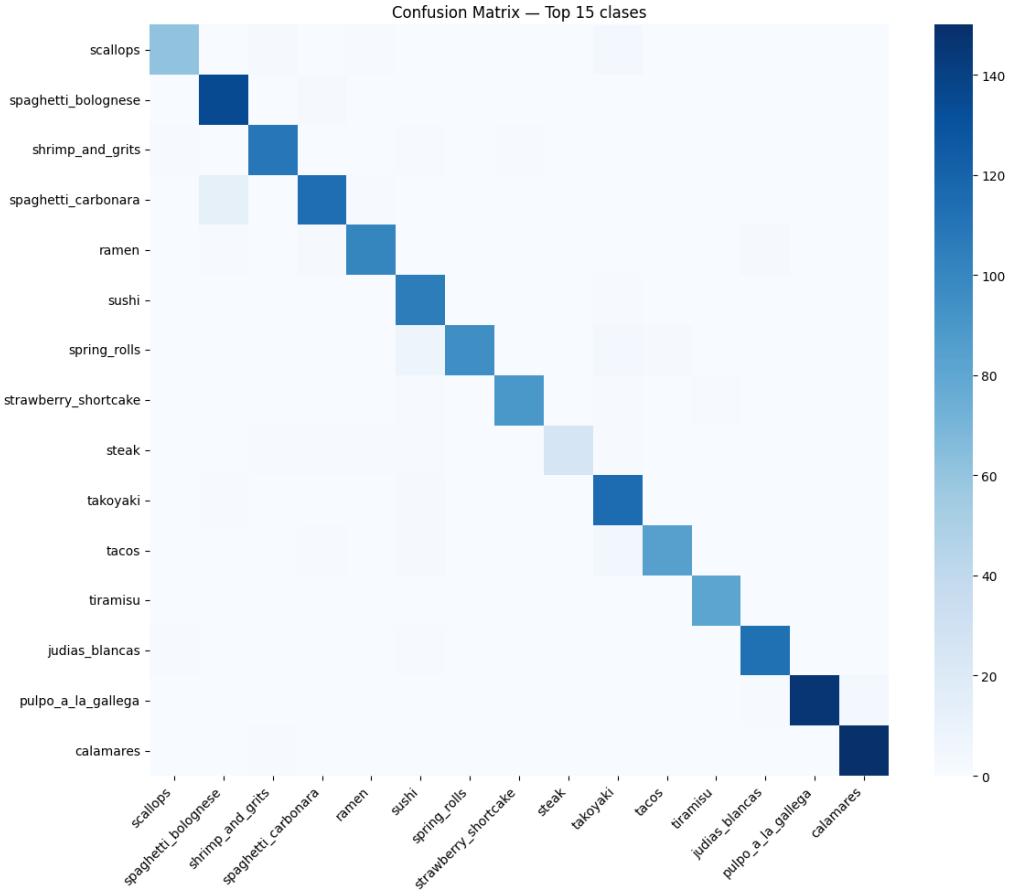
🎯 Total samples evaluados: 17268
1. Top-1 Accuracy : 0.6896
2. Top-3 Accuracy : 0.8414
3. Top-5 Accuracy : 0.8894

📊 Classification Report:

	precision	recall	f1-score	support
albondigas	0.97	0.79	0.87	110
apple_pie	0.54	0.33	0.41	150
baby_back_ribs	0.58	0.73	0.65	150
baklava	0.71	0.69	0.70	150
beef_carpaccio	0.73	0.73	0.73	150
beef_tartare	0.87	0.62	0.72	150
beet_salad	0.57	0.57	0.57	150
beignets	0.81	0.85	0.83	150
bibimbap	0.65	0.89	0.75	150
bocadillo	0.93	0.67	0.78	60
bread_pudding	0.67	0.27	0.39	150
breakfast_burrito	0.75	0.60	0.67	150
bruschetta	0.75	0.49	0.59	150
cachopo	0.95	0.90	0.93	62
caesar_salad	0.87	0.64	0.74	150
calamares	0.94	0.77	0.85	196
cannoli	0.61	0.77	0.68	150
caprese_salad	0.64	0.69	0.66	150
carrilladas	0.00	0.00	0.00	0
carrot_cake	0.67	0.59	0.63	150
ceviche	0.62	0.39	0.48	150
cheese_plate	0.70	0.79	0.75	150
cheesecake	0.52	0.69	0.59	150
chicken_curry	0.72	0.45	0.55	150
chicken_quesadilla	0.63	0.73	0.68	150
chicken_wings	0.52	0.85	0.64	150
chocolate_cake	0.55	0.49	0.52	150
chocolate_mousse	0.48	0.46	0.47	150
chorizo	0.93	0.81	0.86	126
churros	0.88	0.69	0.78	150
clam_chowder	0.85	0.70	0.77	150
club_sandwich	0.88	0.76	0.81	150
cochinillo	0.95	0.72	0.82	88
cocido	0.83	0.91	0.87	100
cordero	0.93	0.83	0.88	81
crab_cakes	0.35	0.72	0.47	150
creme_brulee	0.84	0.75	0.79	150
croque_madame	0.70	0.73	0.71	150
cup_cakes	0.54	0.82	0.65	150
deviled_eggs	0.55	0.91	0.68	150
donuts	0.61	0.61	0.61	150
dumplings	0.92	0.74	0.82	150
edamame	0.95	0.97	0.96	150
eggs_benedict	0.68	0.82	0.74	150
ensalada	0.88	0.83	0.86	103
escargots	0.89	0.57	0.70	150
fabada	0.92	0.85	0.88	144
falafel	0.57	0.73	0.64	150
filet_mignon	0.43	0.53	0.48	150
fish_and_chips	0.78	0.80	0.79	150
foie_gras	0.57	0.39	0.46	150
french_fries	0.88	0.79	0.84	150
french_onion_soup	0.63	0.77	0.70	150

french_toast	0.62	0.64	0.63	150
fried_calamari	0.71	0.63	0.67	150
fried_rice	0.39	0.83	0.53	150
fritura_de_pescado	0.94	0.77	0.85	86
frozen_yogurt	0.72	0.83	0.77	150
garlic_bread	0.83	0.53	0.64	150
gazpacho	0.97	0.95	0.96	80
gnocchi	0.70	0.47	0.56	150
greek_salad	0.65	0.85	0.73	150
grilled_cheese_sandwich	0.54	0.70	0.61	150
grilled_salmon	0.63	0.41	0.50	150
guacamole	0.72	0.87	0.79	150
gyoza	0.61	0.75	0.67	150
hamburger	0.61	0.70	0.65	150
hot_and_sour_soup	0.85	0.80	0.82	150
hot_dog	0.70	0.75	0.73	150
huevos_rancheros	0.57	0.42	0.48	150
hummus	0.70	0.62	0.66	150
ice_cream	0.46	0.75	0.57	150
jamon	0.92	0.76	0.83	79
judias_blancas	0.85	0.73	0.79	154
judias_verdes	0.97	0.83	0.89	146
lasagna	0.71	0.51	0.60	150
lentejas	0.62	1.00	0.76	48
lobster_bisque	0.74	0.84	0.79	150
lobster_roll_sandwich	0.64	0.88	0.74	150
macaroni_and_cheese	0.68	0.57	0.62	150
macarons	0.78	0.79	0.78	150
miso_soup	0.69	0.93	0.79	150
mussels	0.78	0.85	0.82	150
nachos	0.75	0.46	0.57	150
omelette	0.61	0.59	0.60	150
onion_rings	0.63	0.89	0.74	150
oysters	0.91	0.81	0.86	150
pad_thai	0.84	0.75	0.79	150
paella	0.62	0.74	0.67	150
pancakes	0.70	0.73	0.71	150
panna_cotta	0.64	0.55	0.59	150
patatas_bravas	0.98	0.75	0.85	106
peking_duck	0.67	0.67	0.67	150
pho	0.64	0.96	0.77	150
pizza	0.50	0.87	0.64	150
pork_chop	0.44	0.27	0.34	150
poutine	0.84	0.73	0.78	150
prime_rib	0.75	0.73	0.74	150
pulled_pork_sandwich	0.68	0.67	0.68	150
pulpo_a_la_gallega	0.98	0.91	0.95	161
ramen	0.80	0.67	0.73	150
ravioli	0.57	0.34	0.43	150
red_velvet_cake	0.81	0.78	0.80	150
risotto	0.71	0.37	0.49	150
salmorejo	0.94	0.90	0.92	114
samosa	0.78	0.72	0.75	150
sashimi	0.91	0.71	0.79	150
scallops	0.73	0.41	0.52	150
seaweed_salad	0.82	0.86	0.84	150
shrimp_and_grits	0.49	0.73	0.59	150
spaghetti_bolognese	0.79	0.90	0.84	150
spaghetti_carbonara	0.88	0.76	0.81	150
spring_rolls	0.85	0.63	0.73	150
steak	0.58	0.17	0.27	150

strawberry_shortcake	0.73	0.60	0.66	150
sushi	0.51	0.71	0.59	150
tacos	0.54	0.56	0.55	150
takoyaki	0.67	0.77	0.71	150
tiramisu	0.67	0.54	0.60	150
tortilla_de_patata	0.86	0.85	0.86	74
tuna_tartare	0.50	0.45	0.48	150
waffles	0.63	0.64	0.63	150
accuracy			0.69	17268
macro avg	0.71	0.69	0.69	17268
weighted avg	0.71	0.69	0.69	17268



Creación del dataset de test

Propósito: Crear un dataset PyTorch para trabajar cómodamente con imágenes del conjunto de test.

Qué hace:

- Usa `ImageFolder` para cargar imágenes y etiquetas automáticamente.
- Aplica las mismas transformaciones usadas en validación.
- **Verifica el número de imágenes** y clases disponibles.

```
from torchvision.datasets import ImageFolder

test_dataset = ImageFolder(
    root=TEST_DIR,
    transform=transform
)

print(" test_dataset creado")
print("Nº imágenes:", len(test_dataset))
print("Clases:", len(test_dataset.classes))
```

```
✅ test_dataset creado
Nº imágenes: 17268
Clases: 121
```

Predicciones aleatorias con imágenes

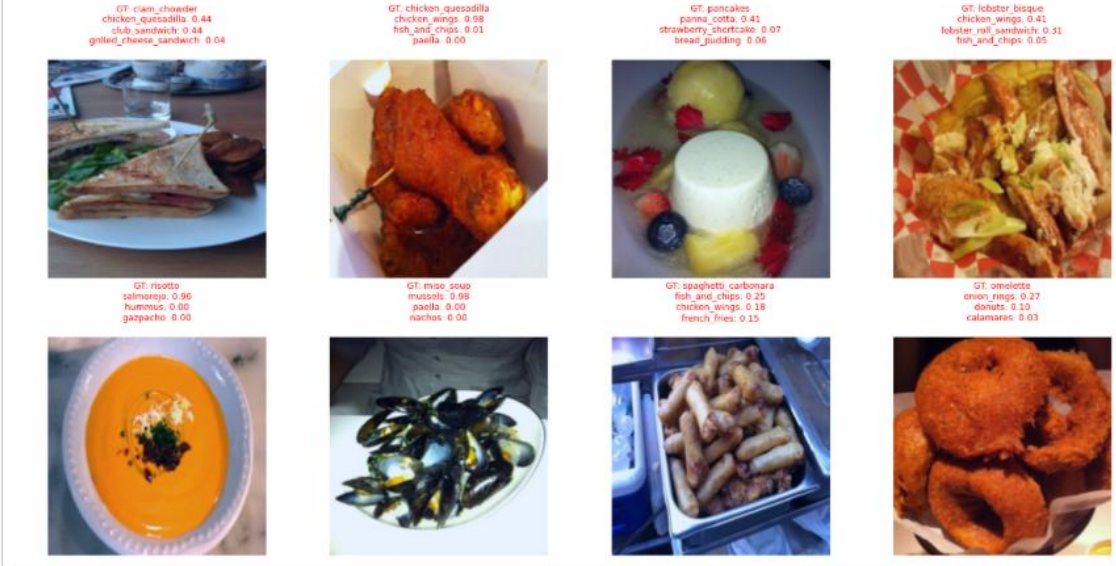
Propósito: Analizar visualmente el comportamiento del modelo sobre imágenes reales.

Qué hace:

- Selecciona **imágenes aleatorias** del conjunto de test.
- Ejecuta **inferencia Top-K** sobre cada imagen.
- **Muestra la imagen**, la **etiqueta real **y las predicciones con probabilidades.
- Colorea el título según si la predicción es **correcta** o **incorrecta**.

```python
import random
import torch
import matplotlib.pyplot as plt
import numpy as np

def mostrar_predicciones_random(
    model, dataset, classes, device,
    n=8, topk=3
):
    model.eval()
    n = min(n, len(dataset))  # seguridad
    idxs = random.sample(range(len(dataset)), n)

    plt.figure(figsize=(16, 8))

    for i, idx in enumerate(idxs):
        img, label = dataset[idx]
        x = img.unsqueeze(0).to(device)

        with torch.no_grad():
            out = model(x)
            probs = torch.softmax(out, dim=1)[0]
            top_probs, top_idxs = probs.topk(topk)

        pred_labels = [classes[j] for j in top_idxs.cpu()]
        pred_probs = top_probs.cpu().numpy()

        ok = top_idxs[0].item() == label
        color = "green" if ok else "red"

        title = f"GT: {classes[label]}\n"
        for p, pr in zip(pred_labels, pred_probs):
            title += f"{p}: {pr:.2f}\n"

        # ---- desnormalizar imagen ----
        img_show = img.permute(1, 2, 0).cpu().numpy()
        img_show = (img_show - img_show.min()) / (img_show.max() - img_show.min())

        plt.subplot(2, (n + 1)//2, i + 1)
        plt.imshow(img_show)
        plt.axis("off")
        plt.title(title, fontsize=9, color=color)

    plt.tight_layout()
    plt.show()
```

```python
mostrar_predicciones_random(
    model=model,
    dataset=test_dataset,
    classes=classes,
    device=DEVICE,
    n=8,
    topk=3
)
```



Matriz de confusión (Top-N clases)

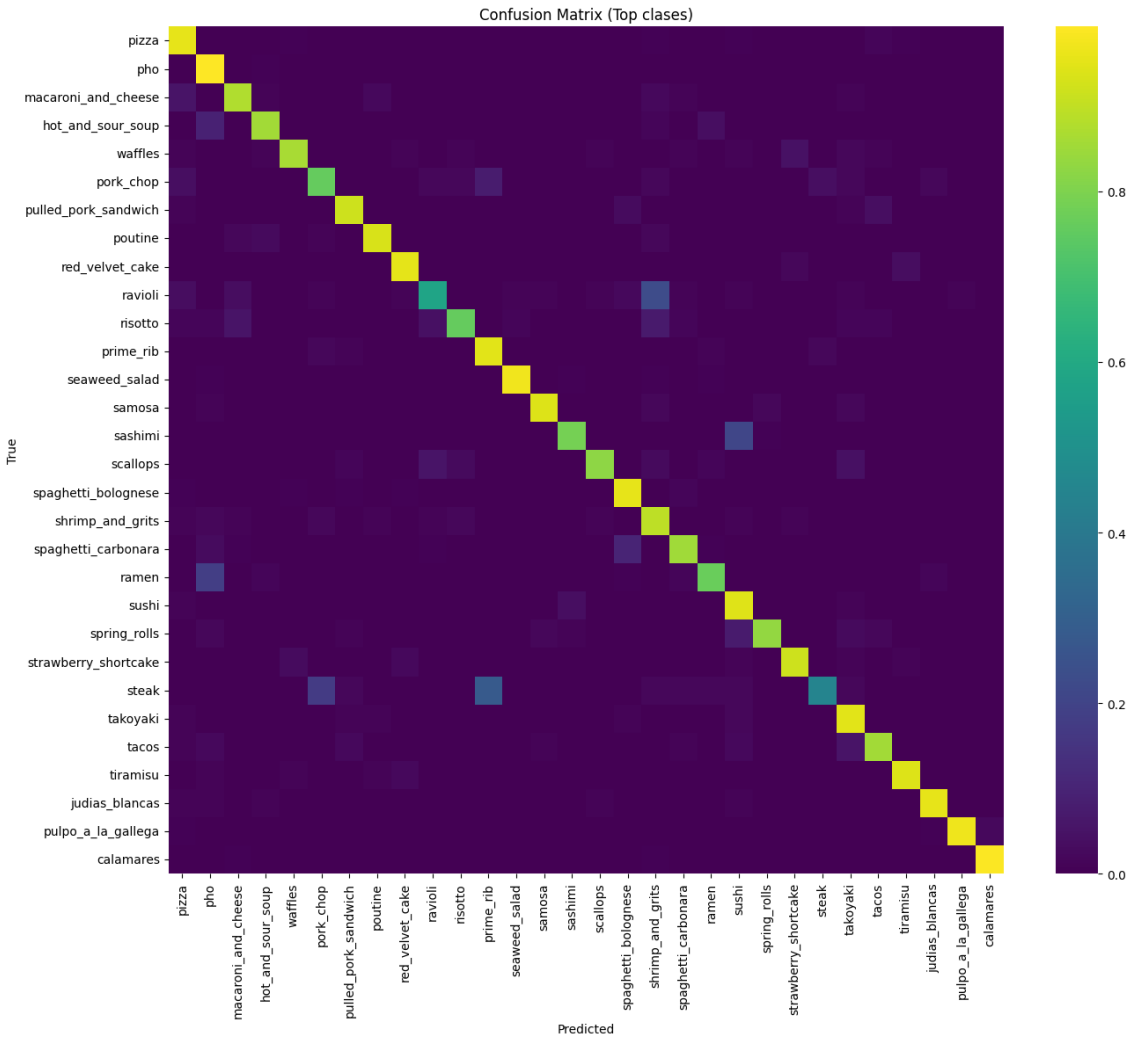
Propósito: Identificar confusiones frecuentes entre las clases más representadas del dataset.

Qué hace:

- Calcula la matriz de confusión global.
- Selecciona las clases con mayor número de muestras.
- Normaliza los valores para facilitar la interpretación.
- Muestra un mapa de calor visual.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

def plot_confusion_top_classes(y_true, y_pred, classes, top_n=30):
    cm = confusion_matrix(y_true, y_pred)
    support = cm.sum(axis=1)

    top_idx = np.argsort(support)[-top_n:]
    cm = cm[top_idx][:, top_idx]
    cls = [classes[i] for i in top_idx]

    cm_norm = cm.astype("float") / cm.sum(axis=1, keepdims=True)

    plt.figure(figsize=(14, 12))
    sns.heatmap(
        cm_norm,
        xticklabels=cls,
        yticklabels=cls,
        cmap="viridis",
        annot=False
    )
    plt.title("Confusion Matrix (Top clases)")
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.tight_layout()
    plt.show()
```

```
plot_confusion_top_classes(y_true, y_pred, classes, top_n=30)
```



Recall por clase

Propósito: Evaluar qué clases son mejor y peor reconocidas por el modelo.

Qué hace:

- Genera el classification report en formato diccionario.
- Extrae el recall de cada clase con soporte válido.
- Ordena las clases por rendimiento.
- Muestra un gráfico de barras horizontal.

Resumen general

Estas celdas cubren **evaluación cuantitativa y cualitativa** completa del sistema, permitiendo:

- Medir rendimiento global.
- Analizar errores por clase.
- Visualizar predicciones reales.
- Tomar decisiones informadas para mejoras futuras.

```python
import pandas as pd
import matplotlib.pyplot as plt

def plot_accuracy_per_class(report):
    df = pd.DataFrame(report).T
    df = df[df["support"] > 0]
    df = df.sort_values("recall")

    plt.figure(figsize=(10, 18))
    plt.barh(df.index, df["recall"])
    plt.xlabel("Recall")
    plt.title("Recall por clase")
    plt.tight_layout()
    plt.show()
```

```python
from sklearn.metrics import classification_report

report_dict = classification_report(
    y_true,
    y_pred,
    target_names=classes,
    output_dict=True,
    zero_division=0
)

print(" report_dict creado")


plot_accuracy_per_class(report_dict)
```

```
 report_dict creado
```