Clase 231 — Capstone 3: visión por computadora con transfer learning

Parte: 8 — Capstones · Fuente: Géron, Hands-On ML 3ª ed. cap. 14-15 + timm + PyTorch Lightning docs. Duración estimada: 180 min.

Clase 231 — Capstone 3: visión por computadora con transfer learning

> *Parte: 8 — Capstones · Fuente: Géron, Hands-On ML 3ª ed. cap. 14-15 + timm + PyTorch Lightning docs. Duración estimada: 180 min.*

Objetivo

Construir un clasificador de imágenes de calidad producción usando transfer learning sobre un backbone moderno (ConvNeXt-tiny / EfficientNetV2-S / ViT-Base/16). Entrenar en dos fases (feature extraction → fine-tuning progresivo), aplicar augmentation moderna (RandAugment, MixUp, CutMix), evaluar con métricas per-clase + slice analysis, y servir vía ONNX + FastAPI con endpoint /predict que recibe imagen en base64. Cerrar el capstone con un fairness check si aplica.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diseñar un pipeline de transfer learning completo: backbone preentrenado en ImageNet → head custom → fine-tuning progresivo con LR diferencial por grupo.
- Aplicar augmentation moderna con Albumentations (RandAugment, MixUp, CutMix, RandomErasing) y verificar invariancia de label.
- Entrenar con PyTorch Lightning + AMP (mixed precision) + torch.compile + grad accumulation, con seed_everything y deterministic algorithms.
- Reportar accuracy + per-class F1 + confusion matrix + slice analysis (Clase 169) y un fairness check (Clase 224) si el dataset tiene atributos sensibles.
- Exportar el modelo a ONNX o TorchScript y servirlo en un endpoint FastAPI /predict que reciba imagen base64.

Fases del capstone

#	Fase	Por qué importa
1	Dataset + EDA	Class imbalance, resolución variable, leak
2	Augmentation	RandAugment + MixUp suben 2-4 puntos de ac
3	Backbone preentrenado	ConvNeXt-tiny / EffNetV2-S / ViT-B/16 vía
4	Fine-tuning progresivo	Feature extraction (head only) → unfreeze
5	Evaluación	Accuracy global engaña con imbalance; per-
6	Serving	Export a ONNX / TorchScript + FastAPI /pre

Definiciones y características

- Transfer learning: reutilizar pesos de un modelo entrenado en una tarea grande (ImageNet, 1.2M imágenes, 1000 clases) como punto de partida para tu tarea chica. Dos modos: feature extraction (congelar backbone, entrenar solo head) y fine-tuning (descongelar progresivamente).

- Backbone moderno (2026): ConvNeXt-tiny (28M params, CNN moderna), EfficientNetV2-S (21M, balance speed/accuracy), ViT-Base/16 (86M, attention puro). Todos disponibles en timm con pesos ImageNet-21k.
- LR diferencial por grupo: el backbone preentrenado necesita LR ~100× menor que el head random. torch.optim.AdamW([{'params': backbone, 'lr': 1e-5}, {'params': head, 'lr': 1e-3}]).
- RandAugment: política automática que aplica N=2 transformaciones random con magnitud M=9 (rotación, color jitter, equalize, shear...). Plug-and-play en torchvision.transforms.v2.
- MixUp / CutMix: combinan 2 imágenes (MixUp = blend pixel-wise α≈0.2; CutMix = pega un parche de B sobre A). Labels se mezclan en la misma proporción. Regularizador potente para datasets chicos.
- AMP (Automatic Mixed Precision): usar float16 para forward/backward, float32 para pesos. 2× speedup y 2× memoria libre en GPUs ≥ Volta. torch.amp.autocast() + GradScaler.
- torch.compile: en torch 2.x, JIT-compila el grafo del modelo. +20-40% throughput sin tocar código. model = torch.compile(model, mode="reduce-overhead").
- Slice analysis: accuracy/F1 desagregado por sub-grupo (tamaño del objeto, iluminación, demografía). Revela disparidades que la métrica global esconde (Clase 169 + 224).

Dataset / recursos

- Opciones de dataset (elegí una):
- Cats vs Dogs (Kaggle, ~800 MB, 25K imágenes, binario) — el clásico para empezar.
- Food-101 subset (10 clases de 101, ~500 MB) — multiclase realista, fondos heterogéneos.
- Plant Disease (PlantVillage, 38 clases, ~1.5 GB) — class imbalance natural, alto impacto agro.
- Casting Defect (Kaggle industrial, binario, ~100 MB) — defect detection, dataset chico (~7K) ideal para mostrar el valor del transfer learning.
- Stack: torch 2.x · torchvision · timm · pytorch-lightning · albumentations · onnx · onnxruntime · fastapi · uvicorn · wandb o mlflow.

Ejercicios

1. Baseline tonto: entrenar una CNN de 2 capas conv from scratch (sin transfer). Reportar accuracy de validation. Esperá <60% en multiclase — establece el piso.
2. Feature extraction: cargar timm.create_model('convnext_tiny', pretrained=True, num_classes=N). Congelar backbone (for p in model.parameters(): p.requires_grad = False), entrenar solo head 5 epochs. Esperá +20-30 puntos sobre el baseline.
3. Fine-tuning progresivo: unfreeze último bloque → 5 epochs con LR 1e-4 → unfreeze full → 10 epochs con LR diferencial (1e-5 backbone, 1e-3 head). Loggear con W&B/MLflow.
4. Augmentation ablation: comparar (a) sin aug, (b) flip + crop, (c) RandAugment, (d) RandAugment + MixUp + CutMix. Reportar curva val_acc.
5. Serving end-to-end: exportar a ONNX (torch.onnx.export), validar con onnxruntime que la inferencia da la misma probabilidad ±1e-4, levantar FastAPI con endpoint POST /predict que reciba {"image_b64": "..."} y devuelva {"class": "...", "prob": 0.94}. Probar con curl.

Homework verificable

Notebook + repo con:

1. Dataset elegido + EDA (distribución de clases, resolución, ejemplos por clase).
2. Pipeline Lightning con 3 fases de entrenamiento (feature extraction → unfreeze parcial → unfreeze

full).

3. Augmentation con Albumentations + verificación de label invariance.
4. Reporte de métricas: accuracy global, per-class F1, confusion matrix, slice analysis sobre al menos 1 dimensión.
5. Si el dataset tiene atributos sensibles (rostros, demografía): fairness check con disparate impact + equal opportunity (Clase 224).
6. Export a ONNX + script serve.py con FastAPI /predict funcionando localmente.
7. README del repo con resultados, decisiones de diseño y limitaciones.

Criterio de aceptación: accuracy de test > 90% en binario o > 75% en multiclase (10+ clases), el endpoint /predict responde en <500 ms en CPU, y el reporte identifica al menos 1 slice donde el modelo subperforma.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Accuracy de train sube a 99%, val se queda	Overfitting clásico. Fix: subir augmentati
CUDA out of memory al hacer fine-tune full	Batch demasiado grande para AMP off. Fix:
Val accuracy es altísima pero el modelo en	Leakage train/test (mismas imágenes duplic
RuntimeError: shape mismatch al exportar a	El head custom espera shape que el dummy_i
Predicciones ONNX difieren de PyTorch en 5	Esperable (precisión float). Fix: validar
Una clase tiene F1=0.0 y nadie lo notó	Class imbalance (esa clase es 2% del datas

Preguntas frecuentes

¿ConvNeXt, EfficientNetV2 o ViT?

Para datasets chicos (<10K imágenes), ConvNeXt-tiny o EfficientNetV2-S ganan: las CNN tienen inductive bias (locality, translation equivariance) que ViT no tiene y por eso ViT necesita 10× más datos para igualar. Para datasets grandes (>100K) o si tenés pretraining en ImageNet-21k, ViT-Base/16 suele ser top. Probá los 3 vía timm y comparalos — son 3 líneas de código cada uno.

¿Feature extraction o fine-tuning?

Feature extraction primero (más rápido, menos riesgo de catastrophic forgetting). Si la accuracy se estanca, hacer fine-tuning progresivo: descongelar el último bloque, después dos, después full, siempre con LR diferencial. Nunca descongelés todo en epoch 1 — el gradiente random del head corrompe los pesos preentrenados.

¿Por qué Lightning y no PyTorch puro?

Lightning te da gratis: AMP, multi-GPU, gradient accumulation, checkpointing, EarlyStopping, logging a W&B/MLflow, y seed_everything. En un capstone querés invertir el tiempo en el modelo, no en el training loop.

¿ONNX o TorchScript para serving?

ONNX si querés portabilidad (servir desde C++, Java, Node, navegador con onnxruntime-web). TorchScript si te quedás en Python/C++ con torch instalado y querés el path más simple. Para FastAPI en producción, ONNX + onnxruntime suele ser 2-3× más rápido que torch eager.

¿Fairness check siempre?

Solo si el dataset tiene atributos sensibles (rostros con edad/género/etnia, datos médicos con demografía). Si clasificás defectos industriales o platos de comida, no aplica. Cuando aplica, es obligatorio — releé Clase 224 antes de publicar el modelo.

Referencias

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow 3ª ed. (O'Reilly, 2022), cap. 14 (CNN) y 15 (transfer learning).
- timm — PyTorch Image Models (Ross Wightman) — 1000+ backbones preentrenados con API única.
- PyTorch Lightning docs — el training loop estándar de facto.
- Albumentations docs — augmentation rápida y composable.
- He, K. et al. Deep Residual Learning for Image Recognition (CVPR 2016). <https://arxiv.org/abs/1512.03385>
- Dosovitskiy, A. et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (ICLR 2021, ViT). <https://arxiv.org/abs/2010.11929>

Siguiente clase

Clase 232 — Portafolio público en GitHub Pages y presentación

Apéndice: notebook (primer bloque)

Pipeline mínimo CPU-friendly sobre patches sintéticos 32×32 (círculo / cuadrado / triángulo). Replica el flujo completo del capstone: baseline → features → augmentation → CNN (opcional torch) → métricas + slice analysis → stubs de transfer learning (timm), export ONNX y serving FastAPI. Requiere: pip install numpy scikit-learn matplotlib. Opcional: torch torchvision timm pytorch-lightning albumentations onnx fastapi.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

rng = np.random.default_rng(42)
SEED = 42
IMG = 32
N = 1000
CLASSES = ['circle', 'square', 'triangle']

def draw_circle(img, cx, cy, r):
    yy, xx = np.ogrid[:IMG, :IMG]
    img[(xx - cx)  2 + (yy - cy)  2 <= r ** 2] = 1.0

def draw_square(img, cx, cy, r):
    img[max(0, cy-r):cy+r, max(0, cx-r):cx+r] = 1.0

def draw_triangle(img, cx, cy, r):
    for dy in range(-r, r+1):
        width = r - abs(dy)
        y = cy + dy
        if 0 <= y < IMG:
            img[y, max(0, cx-width):cx+width+1] = 1.0
```

```
DRAWERS = [draw_circle, draw_square, draw_triangle]

def gen_dataset(n, rng):
    X = np.zeros((n, IMG, IMG), dtype=np.float32)
# ... (truncado)
```

Archivos complementarios

- notebook.ipynb