Clasificación de Personajes de "One Piece" con Deep Learning 🏴‍☠️🤖

Autor: Alejandro Hernández Delgado

Tecnología: PyTorch, Python, Torchvision

Dataset: One Piece Characters (Kaggle)

📋 Descripción del Proyecto

Este proyecto tiene como objetivo desarrollar, entrenar y optimizar una **Red Neuronal Convolucional (CNN)** capaz de identificar y clasificar imágenes de personajes del anime *One Piece*.

El flujo de trabajo abarca desde la construcción de una CNN básica desde cero hasta la implementación de técnicas avanzadas como **Data Augmentation**, **Regularización** y **Transfer Learning** (utilizando EfficientNet-B0) para mejorar la precisión del modelo.

⚙️ Configuración del Entorno y Datos

Requisitos

El proyecto utiliza las siguientes librerías principales:

- `torch` & `torchvision`: Framework de Deep Learning.
- `PIL` (Pillow): Para verificación de integridad de imágenes (limpieza).
- `matplotlib` & `seaborn`: Visualización.
- `kagglehub`: Descarga automática del dataset.

Pipeline de Datos Robusto

Para evitar errores de ejecución y asegurar una comparativa justa, se implementó una estrategia rigurosa:

1. **Limpieza Automática:** Script que ejecuta `img.verify()` para detectar y eliminar imágenes corruptas antes de procesar.
2. **Splitting:** División garantizada en **Train (60%)**, **Validation (20%)** y **Test (20%)**.
3. **Doble Loader:**
    - **Loader Base:** Solo resize/normalización (Para M1, M2, M3).
    - **Loader Aumentado:** Aplica `RandomResizedCrop`, `Flip`, `Rotation` y `ColorJitter` (Exclusivo para M4 y M5).

Metodología y Arquitecturas

Se han diseñado y comparado arquitecturas propias antes de pasar a modelos pre-entrenados:

1. SimpleCNN (Baseline)

Una red secuencial básica para establecer una línea base de rendimiento.

- 3 Bloques Convolucionales.
- Entrenada con **Loader Base** (imágenes estáticas).
- *Problema:* Alta propensión al sobreajuste (overfitting) inmediato.

2. ImprovedCNN (Arquitectura Mejorada)

Una evolución de la red anterior incorporando técnicas de regularización:

- **Mayor profundidad:** 5 bloques convolucionales (hasta 512 filtros).
- **Batch Normalization:** Estabiliza el aprendizaje.
- **Dropout (0.5):** Apagado aleatorio de neuronas.
- **Global Average Pooling:** Reduce parámetros antes del clasificador.

Experimentos y Evolución (M1 - M4)

Se realizó un estudio ablativo incremental para medir el impacto de cada mejora técnica:

ID	Modelo	Técnica Clave	Accuracy (Test)	Observaciones
M1	SimpleCNN	Baseline	*[Result]*	El modelo apenas aprende, sufre overfitting rápido.
M2	ImprovedCNN	+ Arquitectura	*[Result]*	Mejora en estabilidad gracias al Batch Norm y Dropout.
M3	ImprovedCNN	+ Optimizador SGD	*[Result]*	Comparativa de convergencia (Adam vs SGD).
M4	ImprovedCNN	**+ Data Augmentation**	*[Result]*	**Cambio Crítico:** Se usa el Loader Aumentado. Reduce el overfitting drásticamente.

> *Nota: M1, M2 y M3 usan datos limpios. M4 es el primero en ver variaciones de las imágenes, lo que dificulta el entrenamiento pero mejora la generalización.*

🚀 Estrategia Final: Transfer Learning (M5)

Para superar las limitaciones de las redes entrenadas desde cero, se implementó **EfficientNet-B0** pre-entrenada en ImageNet con una estrategia de dos fases.

Pipeline de Entrenamiento

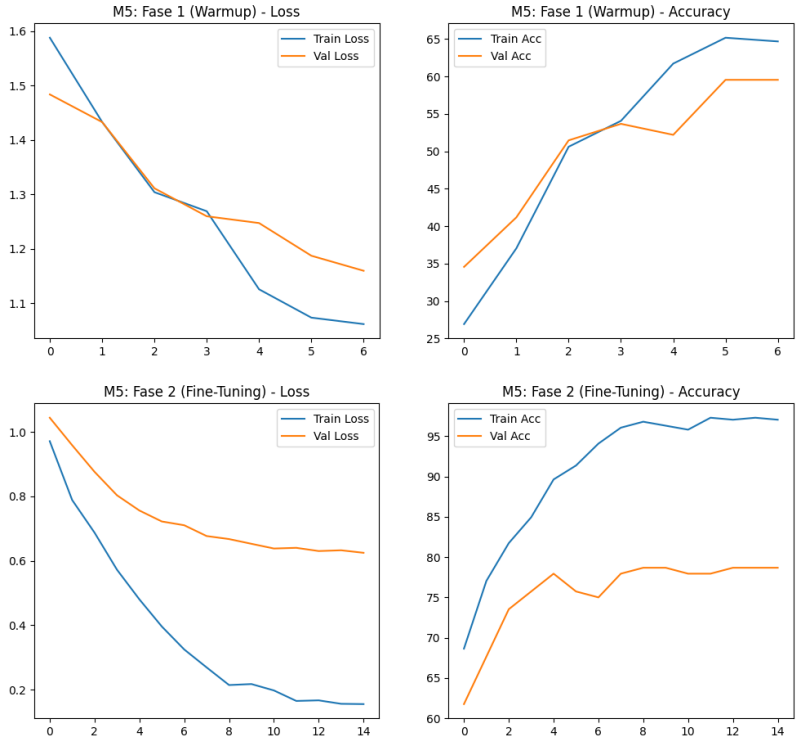
1. **Preprocesamiento:** Uso del **Loader Aumentado** para máxima robustez.
2. **Fase 1 (Warmup):** Se congelan las capas base y se entrena solo el clasificador (Cabecera) por **7 épocas**.
3. **Fase 2 (Fine-Tuning):** Se descongelan todos los pesos y se re-entrena con un **Learning Rate muy bajo (1e-4)** y `weight_decay`.
4. **Scheduler:** Uso de `ReduceLROnPlateau` para reducir el LR si la validación se estanca.

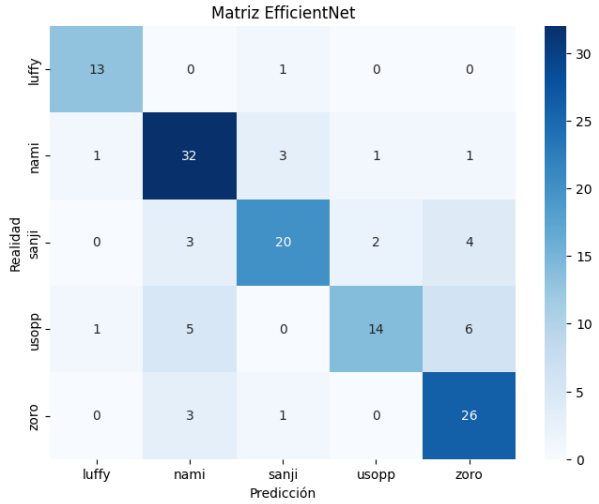
🏆 Resultados Finales

El modelo final (M5) aprovecha el conocimiento previo para adaptarse al estilo "anime":

- **Accuracy en Test: [INSERTAR VALOR FINAL]%**
- **Mejora vs Baseline:** Significativa gracias al Fine-Tuning profundo.

Gráficas y Matrices





🏁 Conclusiones y Trabajo Futuro

Conclusiones

1. **Calidad de Datos:** La implementación del script de limpieza fue fundamental para evitar fallos de lectura (`OSError`) durante el entrenamiento.
2. **Arquitectura vs Datos:** El salto de M2 (Arquitectura) a M4 (Augmentation) demuestra que variar los datos es tan importante como mejorar la red.
3. **El poder de Transfer Learning:** La estrategia de *Warmup + Fine-Tuning* (M5) resultó ser la más efectiva, superando ampliamente a los modelos entrenados desde cero.

Posibles Mejoras (Next Steps)

- **Ensembling:** Combinar predicciones de EfficientNet y ResNet.
- **Test-Time Augmentation (TTA):** Promediar predicciones sobre versiones aumentadas de la imagen de test.
- **Más Datos:** Aumentar el dataset para clases minoritarias.