Sistema de Reconocimiento de Pokémon con Deep Learning

1 de mayo de 2025

Resumen

Este proyecto presenta un sistema de reconocimiento de Pokémon basado en redes neuronales profundas. Utilizando la arquitectura ResNet18 preentrenada en ImageNet, se ajustó la capa final para adaptarse a la clasificación de Pokémon mediante aprendizaje transferido. Se implementó un sistema capaz de entrenar, evaluar y detectar Pokémon en tiempo real utilizando la cámara web.

1. Objetivo

Desarrollar un clasificador de imágenes que identifique correctamente Pokémon a partir de imágenes reales, con soporte para entrenamiento supervisado, partición de datos, visualización de resultados y despliegue funcional en tiempo real.

2. Dataset

El conjunto de datos se generó mediante consultas a la API `PokeAPI`, que proporciona información de Pokémon y enlaces a sus imágenes oficiales. Se almacenó en formato CSV incluyendo nombre, tipo, habilidades, altura, peso, descripción e imagen. Se dividieron los datos en:

- 80 % para entrenamiento (`train.csv`)
- 20 % para prueba (`test.csv`)

3. Modelo y Arquitectura

Se utilizó la red ResNet18 preentrenada. Esta arquitectura incluye conexiones residuales que permiten un entrenamiento profundo sin degradación del gradiente. Se realizó un **fine-tuning** en la última capa:

- Capa modificada: `Linear(512, N)`, donde N es el número de clases (Pokémon).
- Se aplicó **augmentación** de datos: recortes, giros, brillo.
- **Transfer learning** permite reutilizar pesos ya aprendidos.

Fragmento del modelo en PyTorch

```
self.model = models.resnet18(pretrained=True)
self.model.fc = torch.nn.Linear(512, num_classes)
```

Listing 1: Inicialización del modelo con ResNet18

4. Entrenamiento y Evaluación

El entrenamiento se realizó con PyTorch usando entropía cruzada como función de pérdida y optimizador Adam con tasa de aprendizaje 0.001. Se entrenó por 10 épocas, y se guardó el modelo en formato `.pth`.

Código del ciclo de entrenamiento

```
for epoch in range(epochs):
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = self.model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader)
    epoch_accuracy = 100 * correct / total
    losses.append(epoch_loss)
    accuracies.append(epoch_accuracy)
```

Listing 2: Ciclo de entrenamiento del modelo

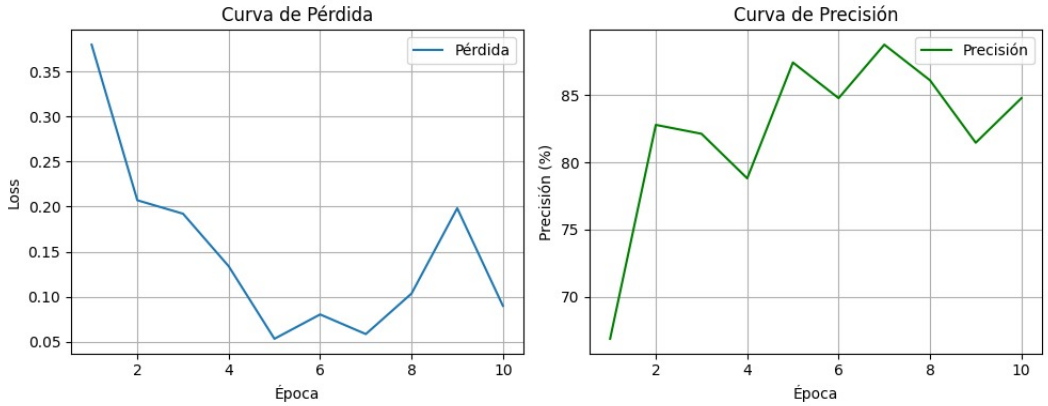

Figura 1: Curvas de pérdida y precisión por época

Curvas de entrenamiento

5. Resultados

El modelo alcanzó una precisión de hasta 85 % en el conjunto de prueba. En pruebas en tiempo real, se logró identificar Pokémon correctamente usando imágenes de la cámara. La augmentación fue clave para evitar sobreajuste dado el tamaño limitado del dataset.

6. Conclusiones

- El aprendizaje transferido permite obtener buenos resultados con pocos datos.
- La separación del dataset mejora la evaluación del modelo.
- Las gráficas de entrenamiento ayudan a diagnosticar sobreajuste y subajuste.
- El sistema es modular y puede escalarse a más clases o modelos más complejos.

Referencias

- He, K., Zhang, X., Ren, S., Sun, J. (2016). *Deep residual learning for image recognition.* In Proceedings of the IEEE conference on CVPR.
- PyTorch documentation: `https://pytorch.org/`
- PokeAPI: `https://pokeapi.co/`