25-5-2025

Reporte

Implementación de Modelo

Leslie del Carmen Sanchez Reyes
TEC DE MONTERREY CAMPUS QUERÉTARO

Contenido

Clasificación de Pokémon mediante Aprendizaje Supervisado con Imágenes

Introducción

En este proyecto, se exploró el diseño, implementación y evaluación de modelos de clasificación de imágenes centrados en personajes de Pokémon, específicamente de la primera generación.
El enfoque se centró en desarrollar un sistema capaz de identificar correctamente una imagen dentro de un conjunto reducido de ocho clases de Pokémon. Dado que los datasets existentes no estaban completamente adaptados al objetivo y tenían un número limitado de ejemplos por clase, fue necesario complementar alguno existente, aplicando procesos de recolección, limpieza y balanceo manual, complementado con técnicas de aumento de datos y preprocesamiento.
A lo largo del proyecto se experimentó con distintas arquitecturas de redes neuronales, desde modelos básicos hasta configuraciones más profundas y optimizadas. Además, se abordaron errores metodológicos y se integraron prácticas de entrenamiento supervisado. Los resultados obtenidos muestran una evolución que si bien, no fue perfecta, ofrece mejoras en el rendimiento del modelo y permiten identificar áreas de oportunidad para trabajos futuros.

Objetivo

Diseñar e implementar un modelo de red neuronal convolucional capaz de clasificar imágenes de Pokémon pertenecientes a 8 clases seleccionadas, mediante la construcción de un dataset personalizado, la aplicación de técnicas de preprocesamiento y aumento de datos, y la evaluación progresiva de distintas arquitecturas de red para optimizar su precisión y capacidad de generalización.

Obtención y Preprocesamiento del Dataset

Con el objetivo de entrenar un modelo de clasificación de imágenes utilizando redes neuronales convolucionales, se construyó un conjunto de datos personalizado tomando como punto de partida el dataset público "7,000 Labeled Pokémon", disponible en la plataforma Kaggle [1]. Este dataset contiene imágenes etiquetadas y

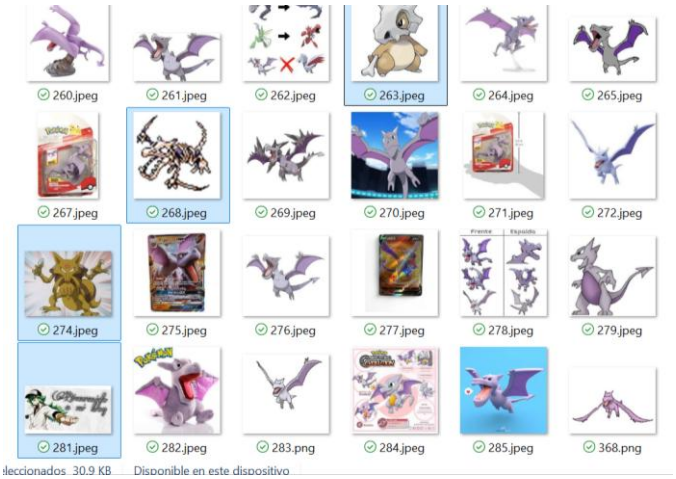
centradas de Pokémon de la primera generación. Contiene 150 folders cada uno con entre 25 a 50 imágenes por Pokémon.
Dado que el volumen de imágenes por clase era insuficiente para entrenar un modelo robusto, se decidió limitar el proyecto a 8 clases específicas de Pokémon, con un total inicial de 3432 imágenes (aproximadamente 429 por clase).

Abra	Aerodactyl	Arbok
Articuno	Bulbasaur	Cubone
Diglett	Ditto	

Se tomó la decisión de aumentar el volumen y la diversidad del conjunto de datos mediante la adquisición de imágenes adicionales. Para ello, se empleó un script existente “downloadImg” [2] para descargar imágenes adicionales desde Google y Pinterest. Este script fue ejecutado para cada una de las ocho clases de Pokémon seleccionadas, generando una nueva colección de imágenes para cada categoría, incrementando el contenido por carpeta hasta aproximadamente 800 imágenes en promedio.
No obstante, este proceso automático trajo consigo una serie de imágenes con problemas, es decir, existían imágenes que no contenían la Pokémon correspondiente o presencia de múltiples personajes que acompañaban al Pokémon objetivo.
Debido a estos problemas, se consideró aplicar una etapa de depuración manual. Este proceso consistió en revisar carpeta por carpeta, imagen por imagen, con el objetivo de garantizar la calidad y precisión del dataset, evaluando que las imágenes obtenidas fueran de acuerdo con lo buscado y en su defecto eliminarlas o recortarlas para enfocarse en el objetivo. La depuración incluía dos acciones principales:

- **Eliminación de imágenes irrelevantes**: aquellas en las que no aparecía claramente el Pokémon de la clase correspondiente fueron descartadas inmediatamente.

- **Recorte manual**: en casos donde el Pokémon aparecía acompañado por otros personajes o en escenas complejas como cuando salía la carta completa del Pokémon, se centró la imagen únicamente en el objetivo de interés recortando el resto de la imagen.



Por ejemplo, en esta imagen que corresponde a la clase de Aerodactyl, están seleccionadas algunas imágenes que no son de valor y que fueron eliminadas.
Una vez concluido el proceso de limpieza y depuración, se procedió a balancear el conjunto de datos. Es importante contar con un dataset balanceado para evitar problemas de clasificación multiclase, ya que un desbalance significativo puede provocar que el modelo aprenda a favorecer las clases con mayor representación, generando sesgos y disminuyendo la generalización.
Para lograr el balance, se identificó la clase con menor número de imágenes válidas tras la depuración, y se utilizó esta cantidad como referencia para todas las demás clases., de modo que todas las clases tuvieran el mismo número de muestras. Así se obtuvo un dataset balanceado, compuesto por 429 imágenes por clase, lo que da un total de 3432 imágenes.

División de los datos: Entrenamiento y Prueba

Con el dataset limpio y balanceado, se procedió a dividirlo de la siguiente forma:

- **80% para entrenamiento.** 2784 imágenes.
- **20% para prueba.** 648 imágenes

Esta distribución fue basada en la división típica sugerida en otros proyectos.
Adicionalmente, se generó un conjunto independiente para validación. Para no afectar el reparto anterior ni reutilizar imágenes ya vistas por el modelo, se descargaron nuevas imágenes adicionales utilizando el mismo procedimiento automatizado. Posteriormente, estas imágenes también fueron depuradas manualmente. El conjunto de validación resultante contiene 25 imágenes por clase (200 imágenes en total). Aunque este tamaño es aún reducido, su expansión se considera como parte del trabajo futuro.

Preprocesamiento de los datos

Para el preprocesamiento de las imágenes como técnica de escalamiento, se consideró la normalización y redimensionamiento. La normalización de las imágenes

con valores de los píxeles de 0-255 a 0-1 con el objetivo de facilitar el proceso de entrenamiento del modelo. La normalización se llevó a cabo tanto para el grupo de train como de test y validation.
El redimensionamiento de las imágenes, por ahora es con un valor de 120x120 píxeles para que todas las imágenes tengan un tamaño estándar.
Además, para mejorar el volumen de los datos se aplicó el Data Augmentation, este aumento de los datos mediante la rotación de las imágenes, desplazamiento horizontal, zoom y volteando las imágenes aleatoriamente. El aumento de datos se realizó inicialmente para los 3 conjuntos de imágenes. Estos cambios los hacemos para que el modelo aprenda a reconocer al mismo Pokémon, aunque este se vea en una posición u orientación diferente. Se utiliza ImageDataGenerator ya que nos ayuda a trabajar con más imágenes, pero cuidado al mismo tiempo el espacio en RAM.

Implementación del Modelo

Primera arquitectura
Como punto de partida se implementó un modelo secuencial sencillo con una sola capa convolucional, seguido por una por una capa de MaxPooling, una capa densa y finalmente una capa de salida softmax con 8 unidades (una por clase). Esta primera arquitectura no se fundamentó bajo ningún paper, sino que se optó por un enfoque más básico con el objetivo de comprender el flujo completo de construcción, entrenamiento y evaluación del modelo, en línea con los contenidos vistos en clase.
Esta arquitectura se presenta a continuación:

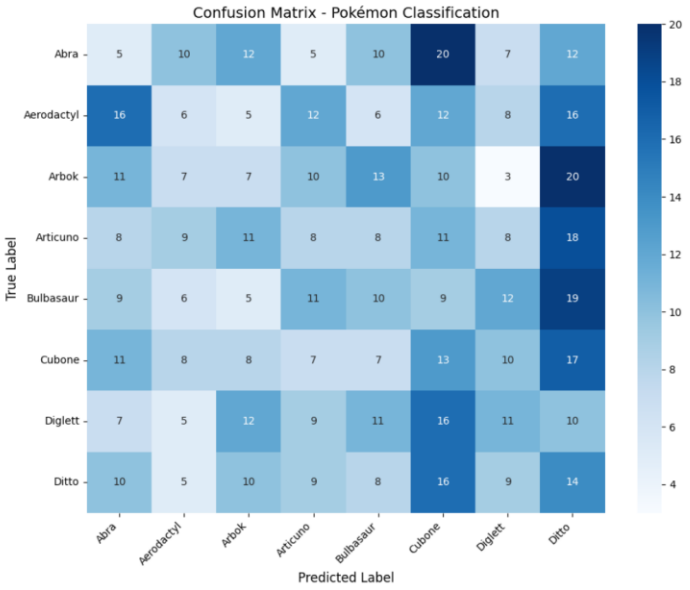
```
model = Sequential([
    Conv2D(10, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2,2),
    Dense(128, activation='relu'),
    Flatten(),
    Dense(8, activation='softmax')
])
```

Esta arquitectura inicial se basó en la experiencia previa de las clases, mediante prueba y error y tomando de referencia trabajos de compañeros de años anteriores. El modelo fue entrenado durante 10 épocas. Los resultados obtenidos fueron:

Accuracy inicial	Loss inicial	Accuracy final	Loss final	Accuracy en test	Loss en test
26.3%	5.27	71.7%	0.87	68.2%	0.98

Se tenía un comportamiento extraño ya que la línea de validación iba por encima de la línea de train, lo cual no es común.
Es claro que los valores en general no son exactamente buenos, sobre todo al revisar la matriz de confusión, el modelo mostraba un desempeño pobre, confundiendo varias

clases entre sí. Esto sugiere que el modelo no estaba aprendiendo representaciones suficientemente discriminativas.



Segunda arquitectura

Para evitar overfitting y reducir el tiempo de entrenamiento, se agregó una capa de Dropout (30%) después de la capa densa. La arquitectura quedó de la siguiente manera:

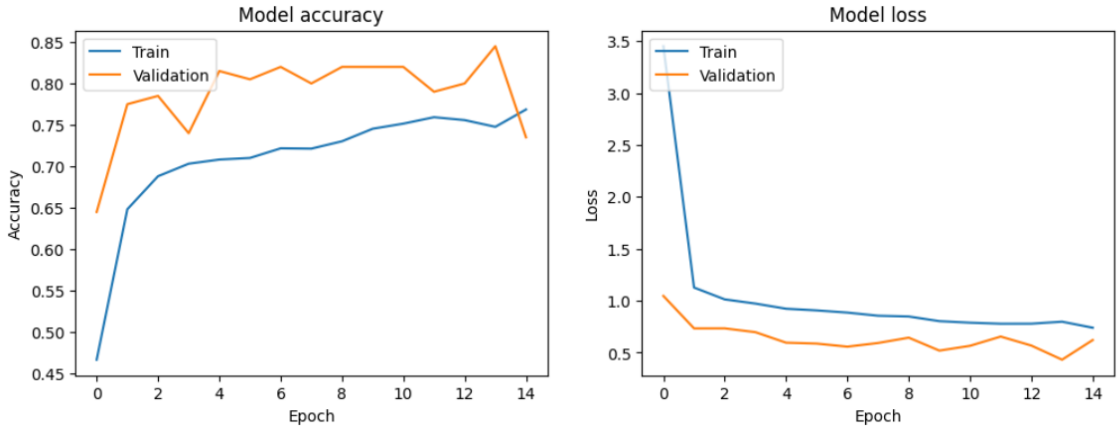
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 10)	280
max_pooling2d (MaxPooling2D)	(None, 74, 74, 10)	0
dense (Dense)	(None, 74, 74, 128)	1,408
dropout (Dropout)	(None, 74, 74, 128)	0
flatten (Flatten)	(None, 700928)	0
dense_1 (Dense)	(None, 8)	5,607,432

Total params: 5,609,120 (21.40 MB)
Trainable params: 5,609,120 (21.40 MB)
Non-trainable params: 0 (0.00 B)

Este nuevo modelo fue entrenado durante 10 épocas. Los resultados obtenidos fueron:

Accuracy inicial	Loss inicial	Accuracy final	Loss final	Accuracy en test	Loss en test
34.9%	4.67	73.7%	0.84	74.5%	0.77



Agregar esta nueva capa significo una mejora en las métricas muy ligera. Sin embargo, al observar el comportamiento de las métricas accuracy y loss por época en train, validation y test mostraban la misma anomalía: las curvas de test y validación estaban por encima de la curva de entrenamiento. Esto podía indicar un posible error en el preprocesamiento. Al hacer análisis de ellos una de las razones que consideramos responsables es haber aplicado data augmentation a los 3 set de datos en lugar de solo haberla aplicado al set de entrenamiento. Estos conjuntos deben representar datos "reales" no vistos y no alterados, para evaluar correctamente el desempeño generalizado del modelo. Se corrigió posteriormente, aplicando data augmentation solo al conjunto de entrenamiento.

Tercera arquitectura

Con el objetivo de mejorar el aprendizaje del modelo, se implementó una nueva arquitectura mejorando la anterior.

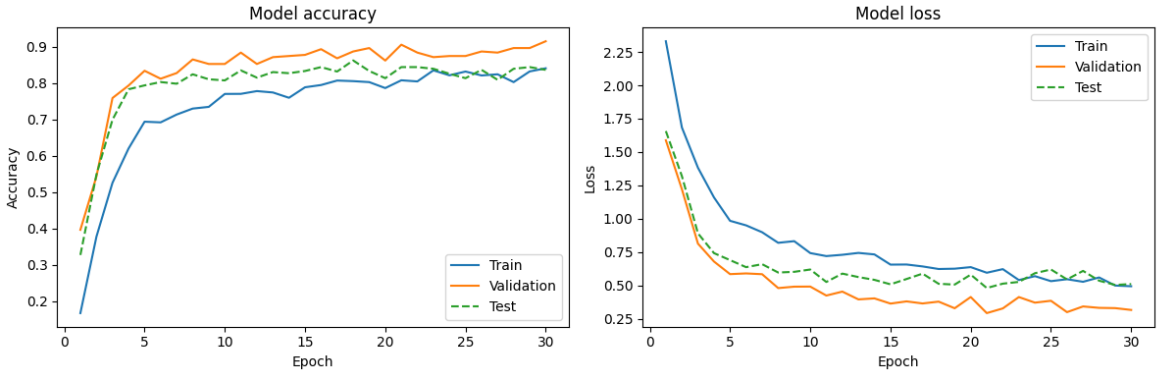
Se añadió un segundo par de capas, una Dense(128) y un Dropout con el 0.1. Lo anterior, para incrementar la capacidad de representación del modelo y procurando que el agregar más capas no sean un motivo de sobreajuste.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 118, 118, 10)	280
max_pooling2d_1 (MaxPooling2D)	(None, 59, 59, 10)	0
dense_3 (Dense)	(None, 59, 59, 128)	1,408
dropout_2 (Dropout)	(None, 59, 59, 128)	0
dense_4 (Dense)	(None, 59, 59, 128)	16,512
dropout_3 (Dropout)	(None, 59, 59, 128)	0
flatten_1 (Flatten)	(None, 445568)	0
dense_5 (Dense)	(None, 8)	3,564,552

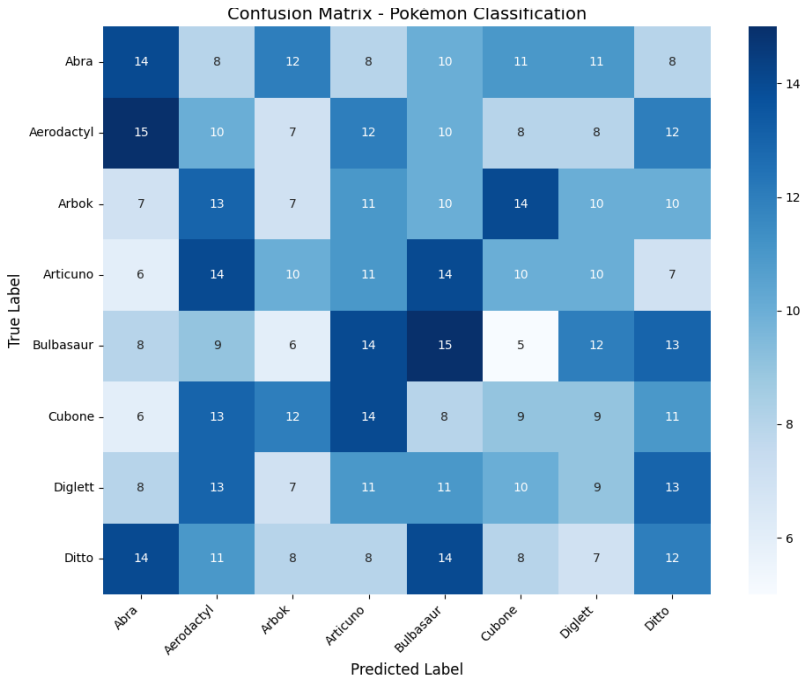
Total params: 3,582,752 (13.67 MB)
Trainable params: 3,582,752 (13.67 MB)
Non-trainable params: 0 (0.00 B)

Esta arquitectura fue entrenada durante **30 épocas**, y se corrigió el preprocesamiento para que solo el conjunto de entrenamiento tuviera data augmentation.
Con los cambios anteriores se obtuvieron los siguientes resultados:

Accuracy inicial	Loss inicial	Accuracy final	Loss final	Accuracy en test	Loss en test	Accuracy inicial
30	0.1678	2.3302	0.8411	0.4945	0.8323	0.5471

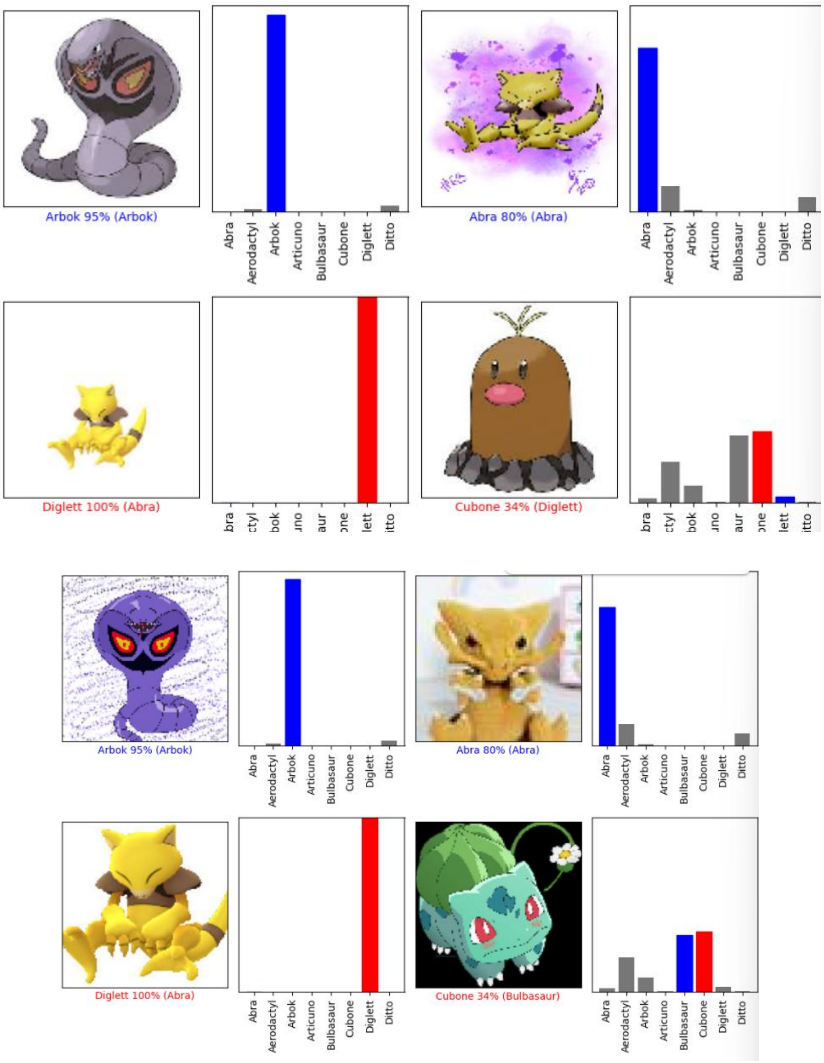


Esta sería la matriz de confusión correspondiente:



Comparando las dos últimas matrices, se puede ver una diferencia, sin embargo, también es notable que sigue teniendo muchos errores al predecir la clase correcta. Y solo para algunos casos la mayoría de las predicciones son acertadas.

Estas imágenes son un ejemplo visual de como es el modelo se ha confundido entre clases.



Conclusión

Este proyecto demostró la viabilidad de construir un modelo de clasificación multiclase de imágenes a partir de un dataset personalizado, utilizando redes neuronales convolucionales como base. El proceso completo, desde la recolección y depuración manual de los datos hasta el ajuste de la arquitectura del modelo, me demostró cómo cada decisión impacta directamente en el rendimiento del modelo.
Inicialmente se partió de una arquitectura básica, sin regularización ni suficiente profundidad, la cual presentó un desempeño limitado con una accuracy en test de solo 68.2% y un loss elevado (0.98). Aunque esta implementación fue útil para entender el flujo general de entrenamiento y evaluación, quedó claro que tenía áreas de oportunidad.

En cada entrenamiento de cada nueva arquitectura era posible ver un progreso, con la primera arquitectura llegamos a un accuracy de 71.7%, y con la última arquitectura mejorada terminamos con un accuracy de 84.1%, lo cual significó más del 10%.

A pesar de las mejoras, es importante mencionar que se identificó que el modelo aún confunde ciertas clases, como se observa en la matriz de confusión. Esto sugiere que los patrones visuales entre algunos Pokémon pueden ser demasiado similares para ser diferenciados por un modelo relativamente simple. El tamaño reducido del conjunto de validación también limita la capacidad para evaluar el rendimiento general del modelo.

Referencias

[1] L. Tian, "7,000 Labeled Pokemon," Kaggle, [Online]. Available: https://www.kaggle.com/datasets/lantian773030/pokemonclassification

[2] O. del Valle Mejía, *QuintupletRecognizer: downloadImg.py*, GitHub repository. [Online]. Available: https://github.com/OsvalDev/QuintupletRecognizer/tree/master/scripts

[3] Y. Tian, "Artificial Intelligence Image Recognition Method Based on Convolutional Neural Network Algorithm," *IEEE Access*, vol. 8, pp. 116388–116397, Jun. 2020, doi: 10.1109/ACCESS.2020.3006097.

[4] P. Mahajan, “Understanding residual network (ResNet) architecture implementation in PyTorch,” Medium, Sep. 9, 2020. [Online]. Available: https://medium.com/analytics-vidhya/understanding-resnet-architecture-869915cc2a98