Primer Taller IA

Experimentos, arquitecturas, resultados y conclusiones

- Jojan Daniel Aguirre
- Juan Manuel Marín
- David Henao
- Óscar Andrés Gómez

Objetivo

- Comparar implementaciones **manuales** (desde cero) vs **TensorFlow/Keras**
- Medir desempeño en:
  - **Flores** (multiclase, 10 clases)
  - **Pokémon** (binario)
- Analizar el impacto de:
  - Regularización **L2**
  - **Data augmentation**
  - Agregar **capas ocultas**

Datasets (resumen)

Flores (multiclase)

- Imágenes: $64 \times 64 \times 3$
- Train: **168** | Test: **42**
- Features aplanadas: **12288**
- Clases: **10** (0–9)

Pokémon (binario)

- Imágenes: $64 \times 64 \times 3$
- Train: **373** | Test: **78**
- Métricas extra: **precision, recall, F1** (por posible desbalance)

Flores (multiclase)

Tamaño e integridad

- CSV: **210** filas | Imágenes presentes: **210** | Faltantes: **0**
- Clases: **10** (0–9)

Distribución por clase (n)

Clase	0	1	2	3	4	5	6	7	8	9
#	21	20	19	22	21	25	23	15	26	18

Características de imagen (subset 200 imgs)

- Modo: **RGBA** (tienen canal alpha)
- Tamaño más común: **128×128** (199/200); hay un outlier **208×208** (1/200)

Stats de pixeles (subset, resize 64×64, normalizado [0,1])

Pokémon (binario)

Tamaño e integridad

- Train: CSV **373** | Imágenes presentes: **373** | Faltantes: **0**
- Test: CSV **78** | Imágenes presentes: **78** | Faltantes: **0**

Balance de clases (0/1)

Split	Clase 0	Clase 1
Train	123	250
Test	28	50

Características de imagen

- Modo: **RGB**
- Tamaño: **256×256** (train/test)

Lectura rápida del EDA (implicaciones)

- **Flores**: dataset pequeño (210 imgs) + multiclase + canal alpha (RGBA) → alto riesgo de sobreajuste al aplanar.
- **Pokémon**: imágenes homogéneas (RGB 256×256) y separación train/test ya definida; hay **desbalance** hacia clase 1 (≈2:1).
- Las medias/std train vs test en Pokémon son muy parecidas → no se ve shift fuerte de iluminación global.

Parte A — Flores (Multiclase)

Experimento A1 — Manual One-vs-Rest (OVR) con Sigmoid

Arquitectura

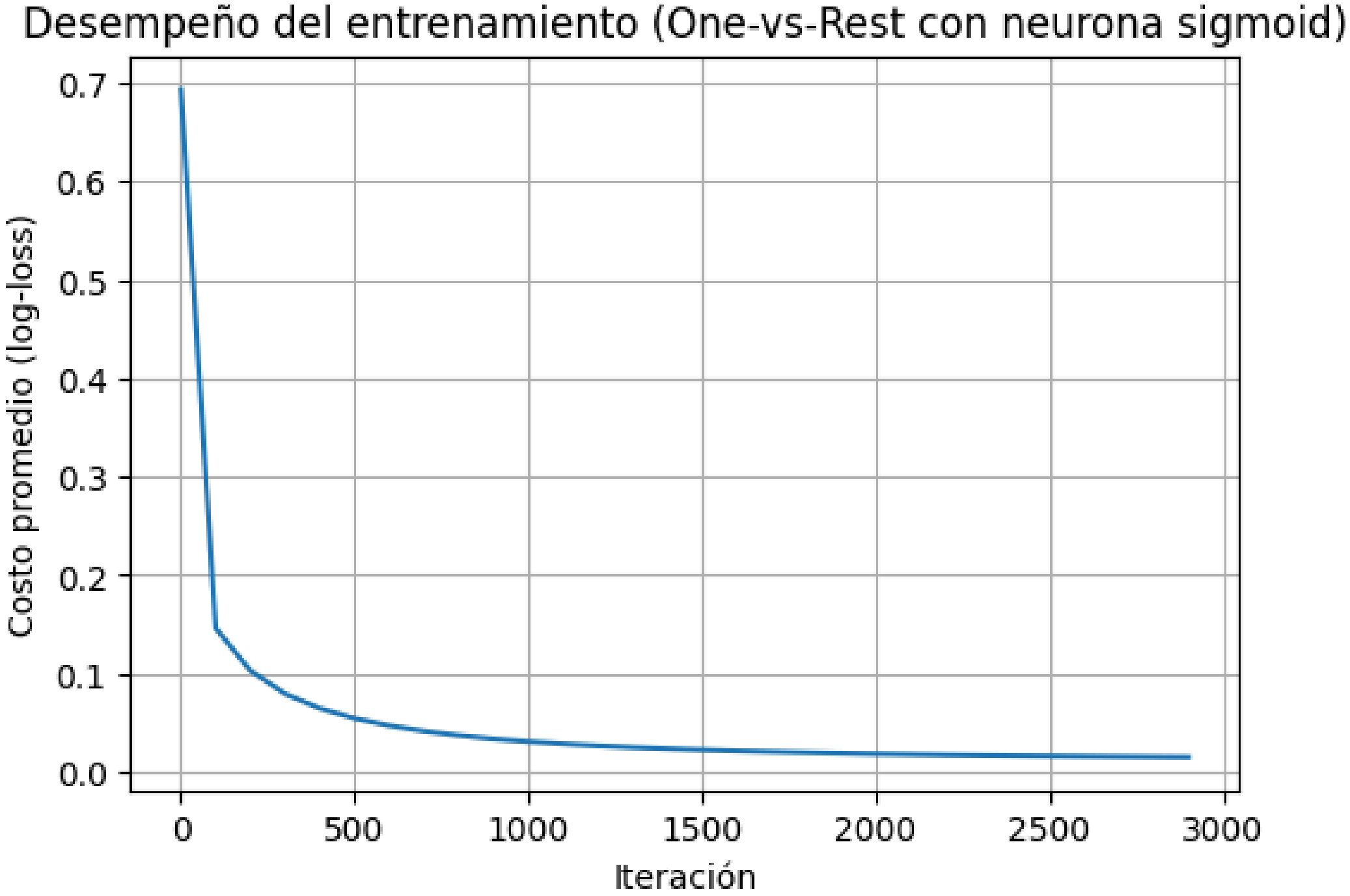
- Entrada: vector de **12288** features (imagen aplanada)
- Modelo: K clasificadores binarios independientes
  - Para cada clase k: Regresión logística (sigmoid)
- Regularización: **L2** (`lambda_=1.0`)
- Predicción: `argmax` sobre $P_k(x)$

Entrenamiento

- Descenso por gradiente (3000 iteraciones)
- `learning_rate=0.005`

Experimento A1 — Resultados

- **Train accuracy**: **100.00%**
- **Test accuracy**: **35.71%**
- Sobreajuste fuerte: el modelo memoriza train (dataset pequeño, espacio de features grande).


Desempeño del entrenamiento (One-vs-Rest con neurona sigmoid)
Costo promedio (log-loss)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
500
1000
1500
2000
2500
3000
Iteración

Experimento A2 — Manual Softmax

Arquitectura

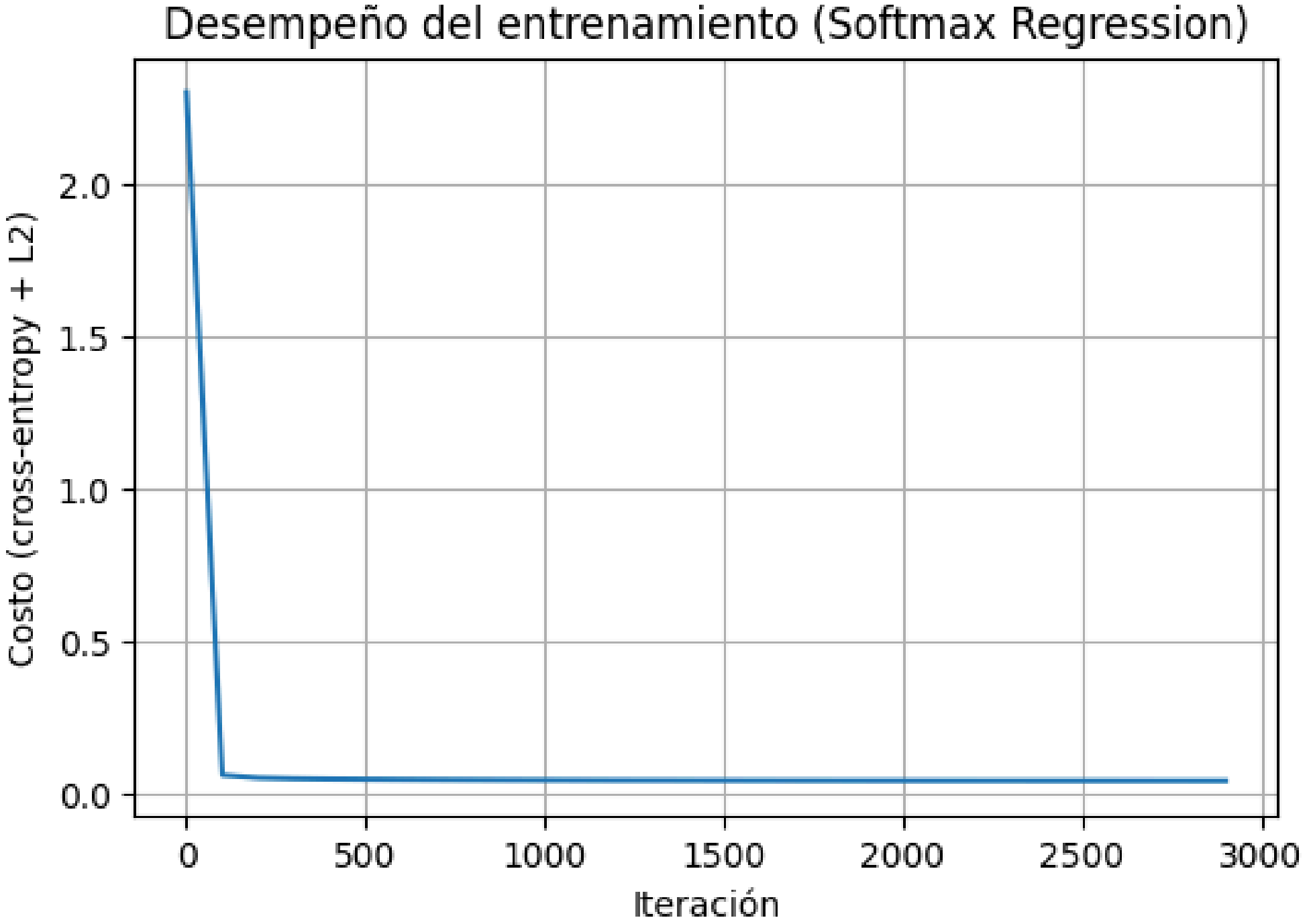
- Entrada: vector de **12288** features
- Modelo: **una sola capa lineal multiclase**
  - $Z = WX + b$ (K salidas)
  - Activación: **softmax** (estable numéricamente con log-sum-exp)
- Pérdida: cross-entropy
- Regularización: **L2** (`lambda_=1.0`)

Entrenamiento

- Descenso por gradiente (3000 iteraciones)
- `learning_rate=0.05`

Experimento A2 — Resultados

- **Train accuracy**: **100.00%**
- **Test accuracy**: **35.71%**
- Mismo techo en test que OVR: sigue siendo un clasificador lineal sobre features aplanadas.


Desempeño del entrenamiento (Softmax Regression)
Costo (cross-entropy + L2)
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000
Iteración

Experimento A3 — Keras baseline (Softmax lineal)

Arquitectura

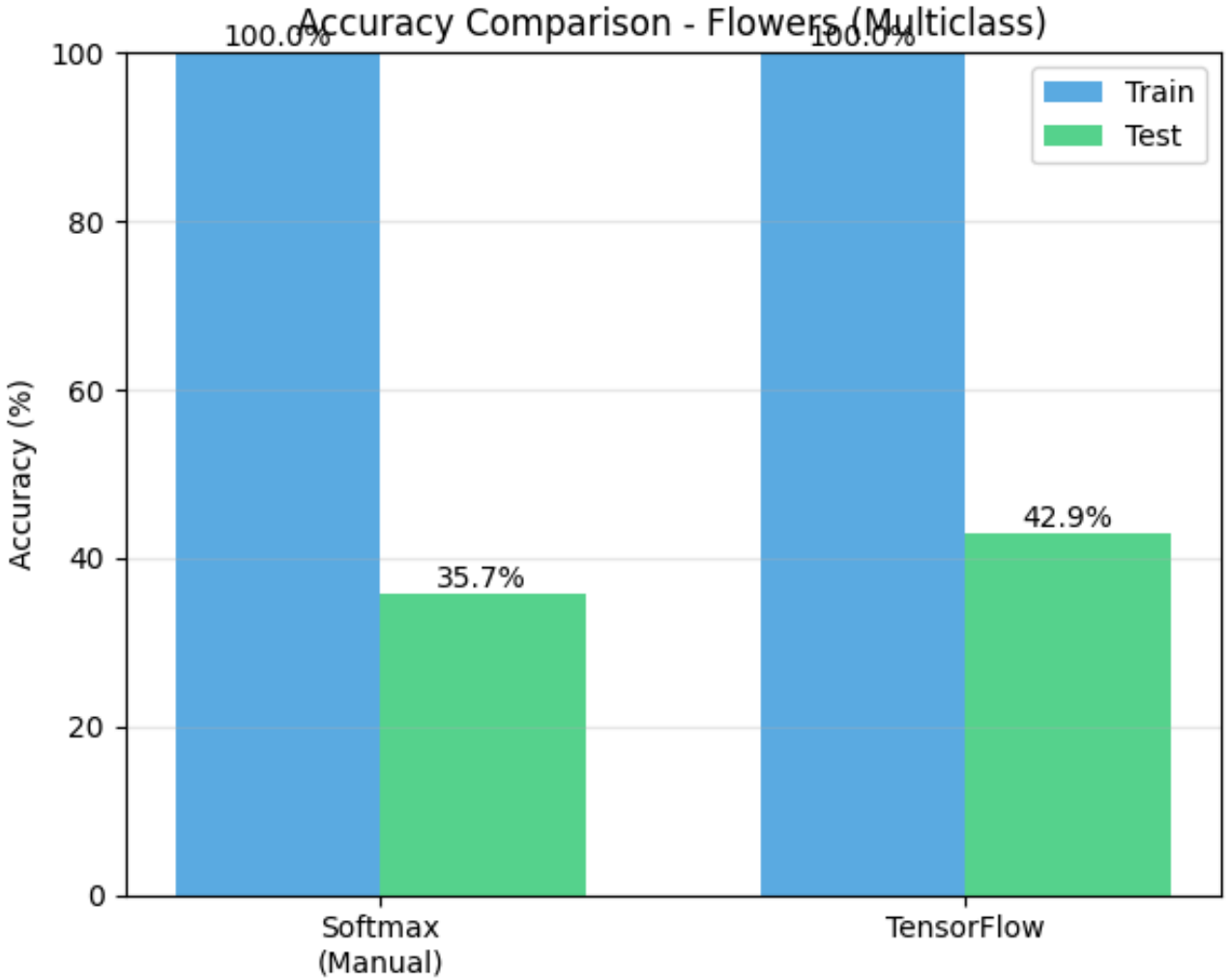
- `Sequential([ Dense(K, activation='softmax', kernel_regularizer=l2(0.01)) ])`
- Nota: equivalente a softmax lineal (sin capas ocultas)

Entrenamiento

- Optimizador: `SGD(lr=0.05)`
- `epochs=30`, `batch_size=32`
- Loss: `sparse_categorical_crossentropy`

Experimento A3 — Resultados

- **Train accuracy**: **100.00%**
- **Test accuracy**: **42.86%**
- Mejora vs manual (35.71% → 42.86%), pero aún limitado por ser lineal.


Accuracy Comparison - Flowers (Multiclass)
Accuracy (%)
100
80
60
40
20
0
100.0%
35.7%
100.0%
42.9%
Train
Test
Softmax
(Manual)
TensorFlow


Training Loss Comparison
Loss
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
3000
Iterations
Softmax (Manual)
TensorFlow

Experimento A4 — Keras + Data Augmentation (sin capa oculta)

Augmentación (solo train)

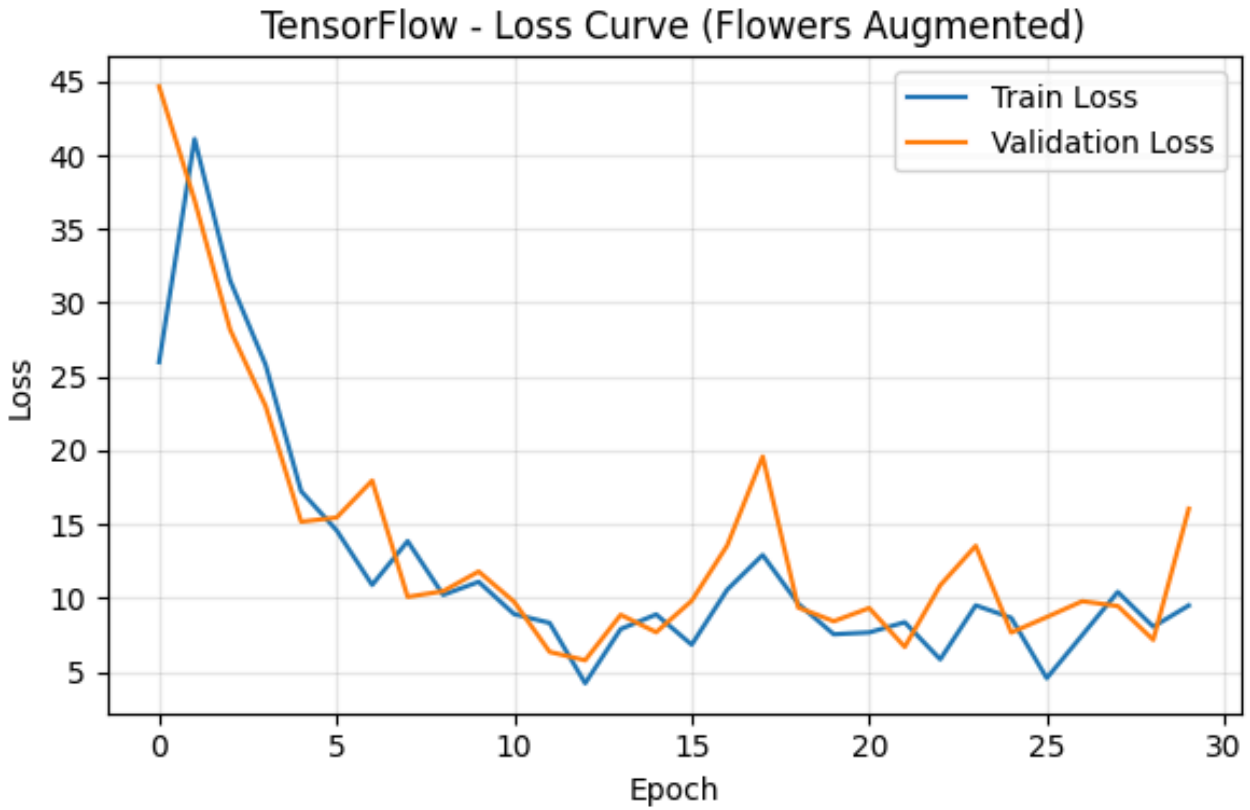
- `rotation_range=20`
- `zoom_range=0.15`
- `horizontal_flip=True`

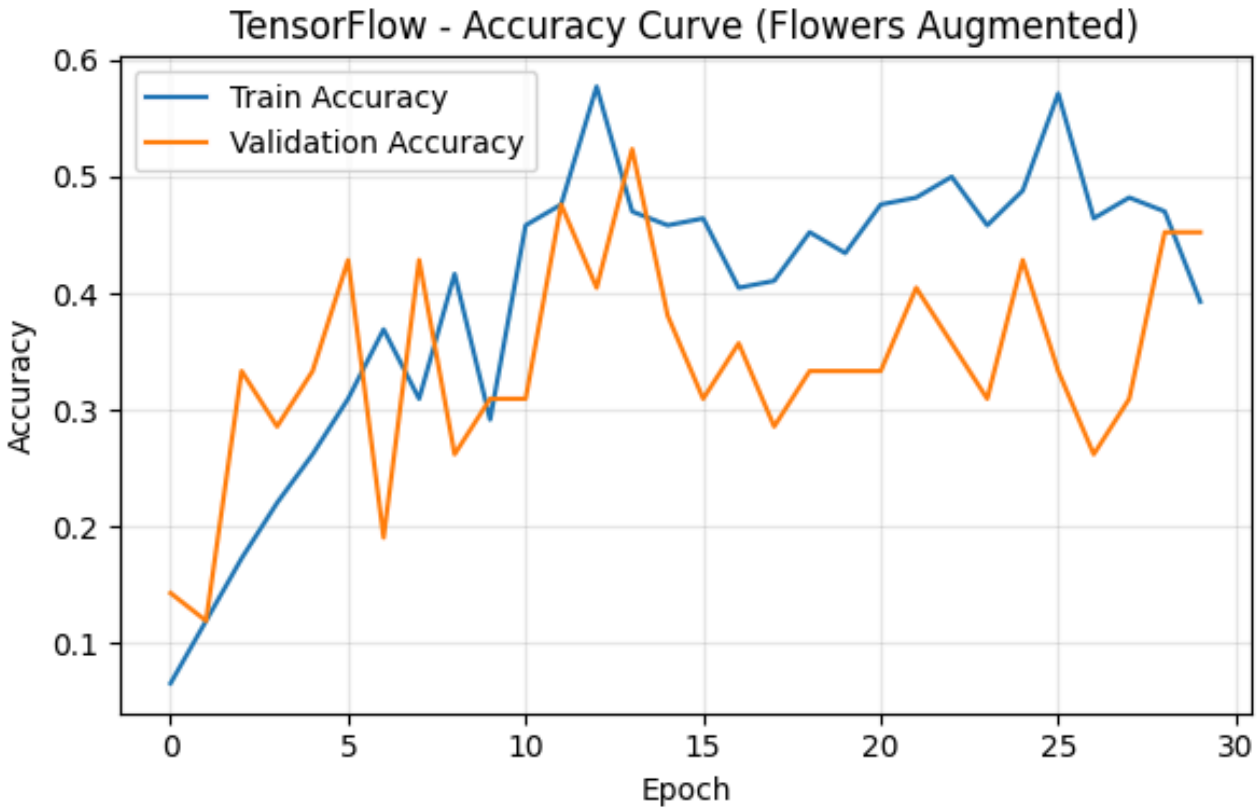
Arquitectura

- `Input(64,64,3) -> Flatten -> Dense(K, softmax)`
- Regularización: L2 `l2(0.01)`

Experimento A4 — Resultados

- (sin aug) **Test**: **42.86%**
- (con aug) **Train**: **48.21%**
- (con aug) **Test**: **45.24%**
- Ligera mejora en test, y caída grande en train: el modelo deja de memorizar ejemplos exactos.


TensorFlow - Loss Curve (Flowers Augmented)
Train Loss
Validation Loss
Loss
Epoch
45
40
35
30
25
20
15
10
5
0
5
10
15
20
25
30


TensorFlow - Accuracy Curve (Flowers Augmented)
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20
25
30

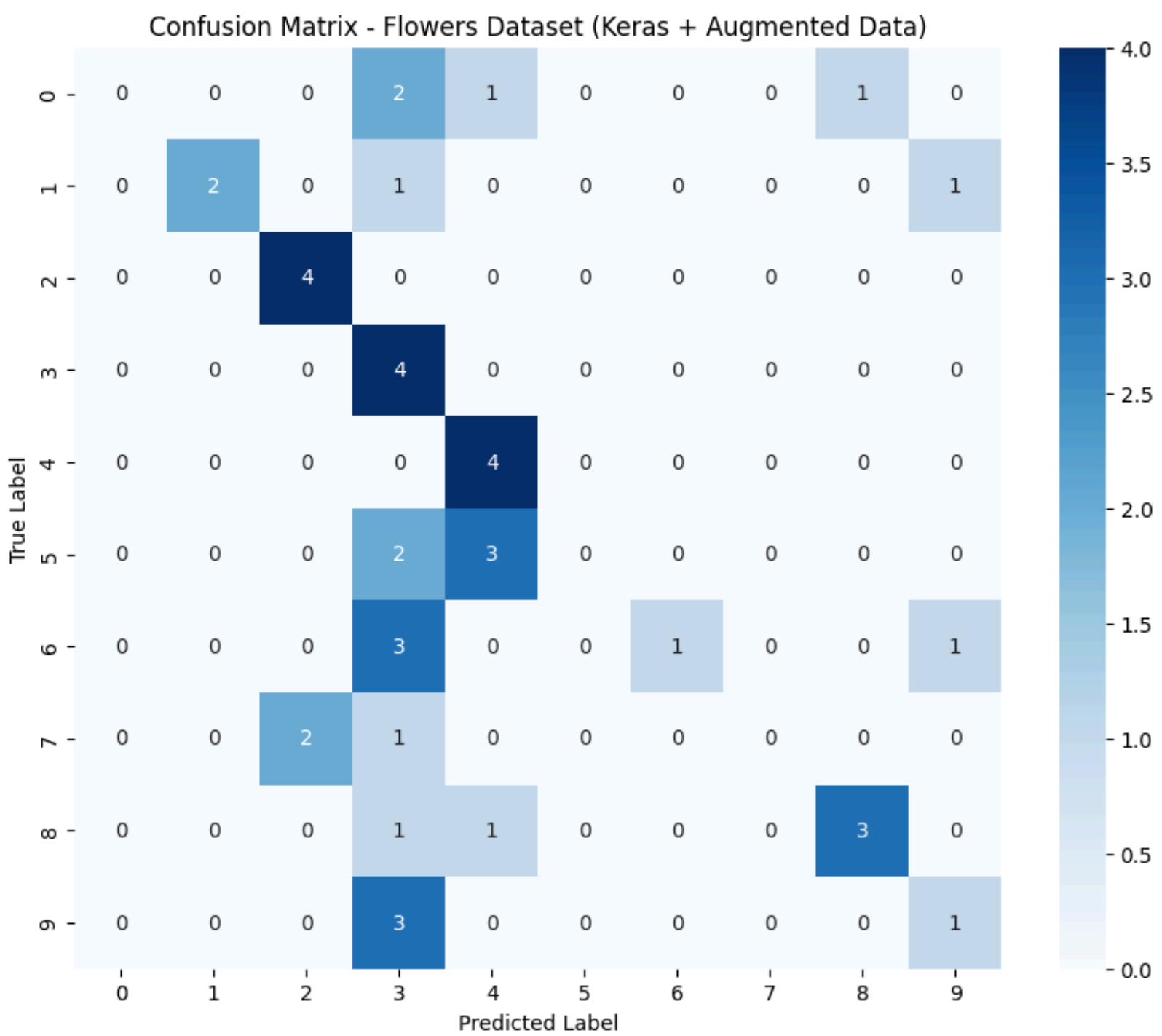
Experimento A5 — Matriz de confusión (Flores, Keras+Aug)

Qué se observó

- Baja precisión global consistente con **45.24%** en test.
- Confusión marcada entre clases; predicciones se concentran en algunas clases (p.ej. **3** y **4**).

Implicación

- El modelo **Flatten+Dense** no aprende rasgos espaciales discriminativos.
- Recomendación natural: usar **CNNs** (Conv2D) o transferencia (MobileNet/ResNet).


Confusion Matrix - Flowers Dataset (Keras + Augmented Data)
True Label
Predicted Label
0 0 0 2 1 0 0 0 1 0
0 2 0 1 0 0 0 0 0 1
0 0 4 0 0 0 0 0 0 0
0 0 0 4 0 0 0 0 0 0
0 0 0 0 4 0 0 0 0 0
0 0 0 2 3 0 0 0 0 0
0 0 0 3 0 0 1 0 0 1
0 0 2 1 0 0 0 0 0 0
0 0 0 1 1 0 0 0 3 0
0 0 0 3 0 0 0 0 0 1
0 1 2 3 4 5 6 7 8 9
4.0 3.5 3.0 2.5 2.0 1.5 1.0 0.5 0.0

Experimento A6 — Keras + Aug + 1 capa oculta

Arquitectura

- `Input(64,64,3) -> Flatten -> Dense(128, relu) -> Dense(K, softmax)`

Intención

- Introducir no-linealidad para capturar patrones más complejos.

Experimento A6 — Resultados y lectura

- (con aug, sin capa oculta) **Test: 45.24%**
- (con aug + Dense(128,relu)) **Test: 28.57%**

Lectura rápida

- Empeora en test: sugiere que esta configuración (tasa de aprendizaje/regularización/capacidad) no está bien calibrada para el dataset.
- Conclusión práctica: para imágenes, un incremento en la complejidad del modelo no implica una mejora en su desempeño.

Parte B — Pokémon (Binario)

Experimento B1 — Manual Sigmoid

Arquitectura

- Entrada: vector de features aplanadas
- Modelo: $\hat{y} = \sigma(w^T x + b)$
- Regularización: **L2** (`lambda_=1.0`)

Entrenamiento

- Descenso por gradiente (3000 iteraciones)
- `learning_rate=0.005`

Experimento B1 — Resultados

- **Train accuracy**: **99.73%**
- **Test accuracy**: **91.03%**
- **F1 (test)**: **0.9346**
- Muy buen desempeño para un modelo lineal: el problema parece más separable (o el dataset menos complejo).

Experimento B2 — Keras baseline (sigmoid lineal)

Arquitectura

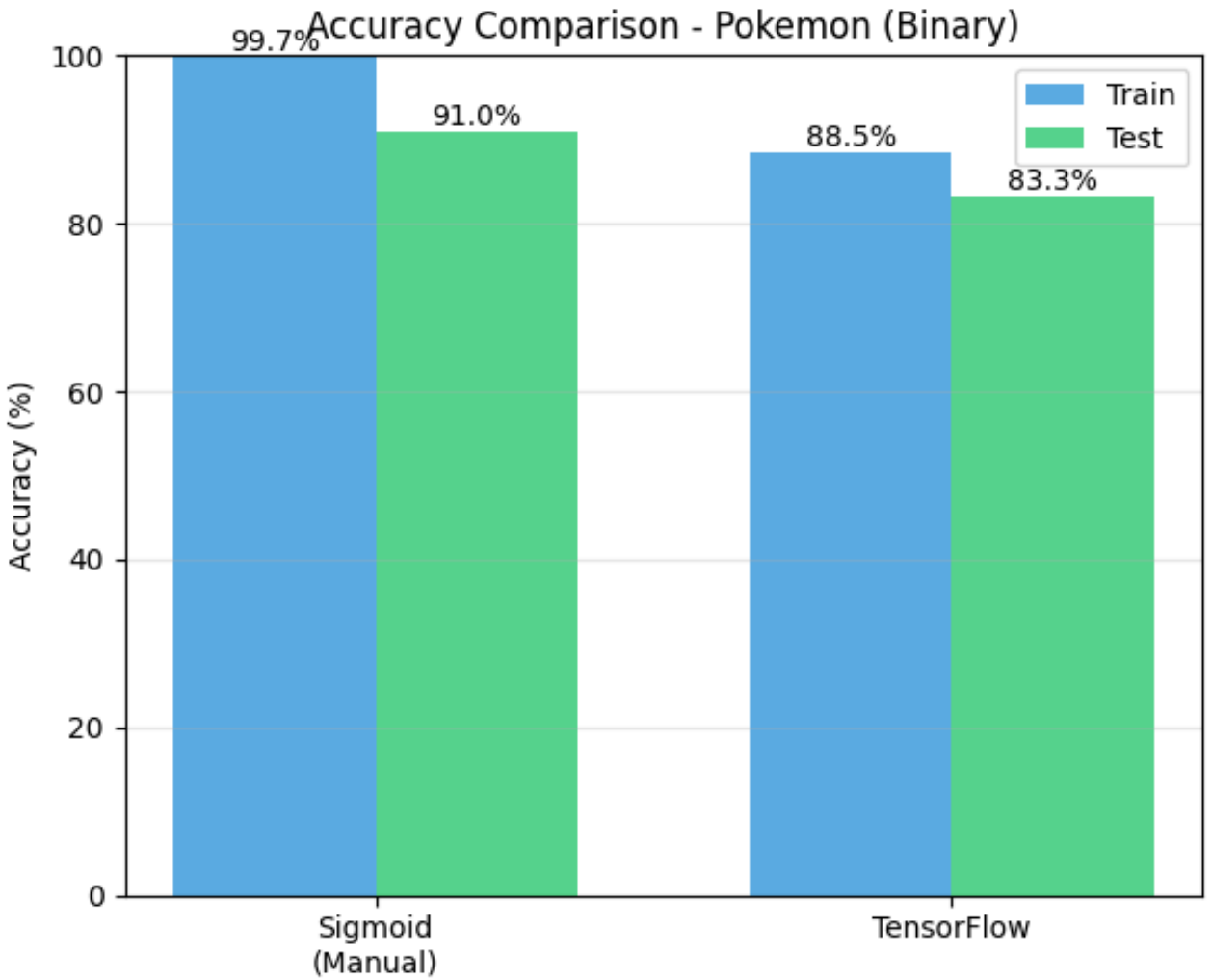
- `Sequential([ Dense(1, sigmoid, kernel_regularizer=l2(0.01)) ])`

Entrenamiento

- `SGD(lr=0.005)`, `binary_crossentropy`
- Métricas: accuracy, precision, recall

Experimento B2 — Resultados

- **Train accuracy**: **88.47%**
- **Test accuracy**: **83.33%**
- **F1 (test)**: **0.8738**
- Menor que el manual; puede depender de detalles de preprocesamiento/regularización/entrenamiento.


Accuracy Comparison - Pokemon (Binary)
Accuracy (%)
100
80
60
40
20
0
99.7%
91.0%
88.5%
83.3%
Train
Test
Sigmoid
(Manual)
TensorFlow


Training Loss Comparison
Loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
500
1000
1500
2000
2500
3000
Iterations
Sigmoid (Manual)
TensorFlow

Experimento B3 — Keras + Data Augmentation (sin capa oculta)

Augmentación

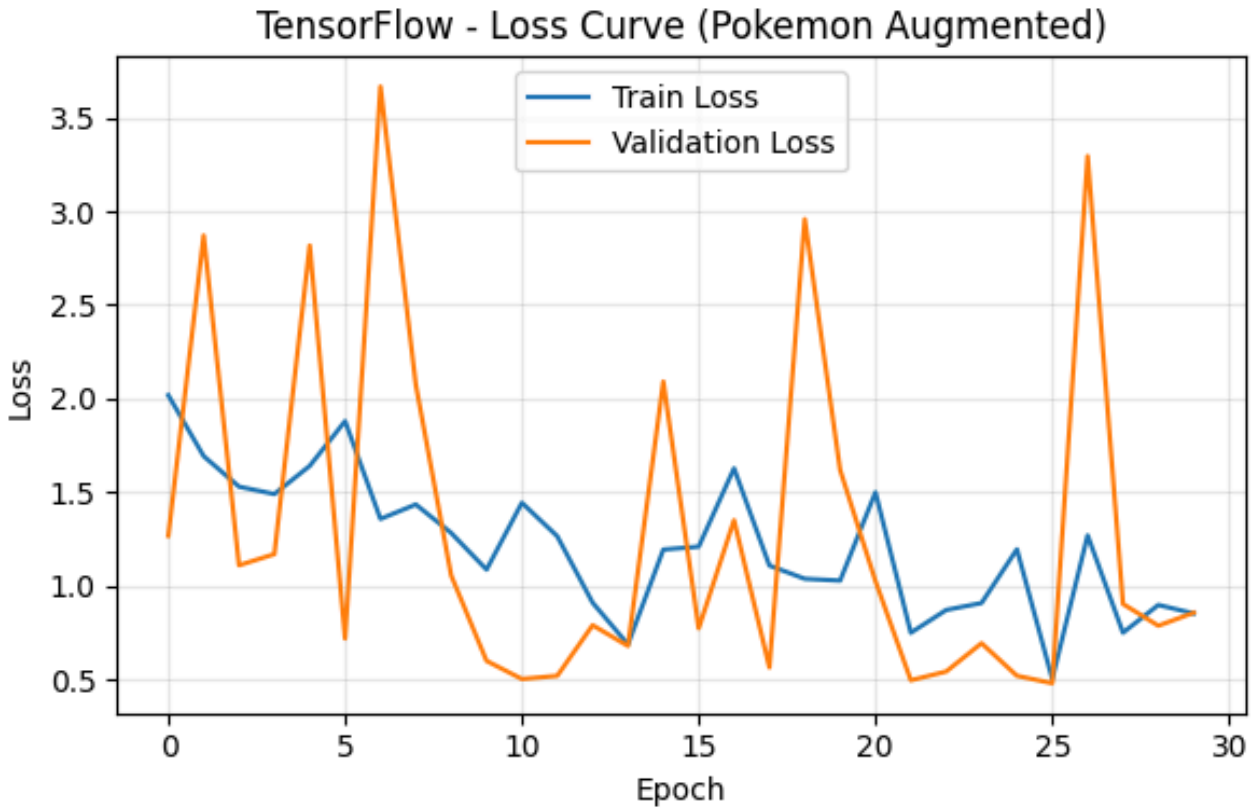
- `rotation_range=20`, `zoom_range=0.15`, `horizontal_flip=True`

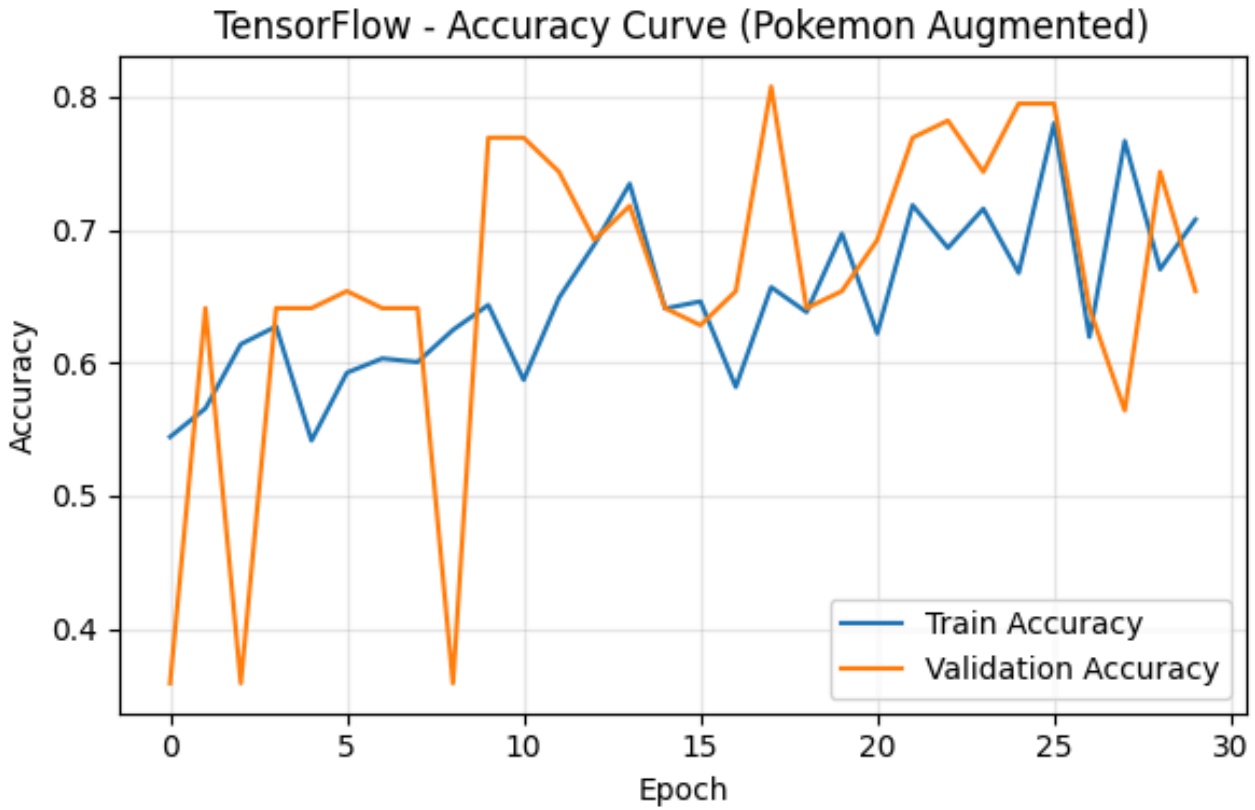
Arquitectura

- `Input -> Flatten -> Dense(1, sigmoid)`

Experimento B3 — Resultados y lectura

- (sin aug) **Test**: **83.33%**, **F1**: **0.8738**
- (con aug) **Train**: **52.55%**
- (con aug) **Test**: **65.38%**, **F1**: **0.6301**
- Augmentación *perjudica* con este modelo: probablemente genera variaciones que el modelo lineal no puede absorber (y/o hiperparámetros no ajustados).


TensorFlow - Loss Curve (Pokemon Augmented)
Train Loss
Validation Loss
Loss
Epoch
0
5
10
15
20
25
30
0.5
1.0
1.5
2.0
2.5
3.0
3.5


TensorFlow - Accuracy Curve (Pokemon Augmented)
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0
5
10
15
20
25
30
0.4
0.5
0.6
0.7
0.8

Experimento B4 — Keras + Aug + 1 capa oculta

Arquitectura

- `Input -> Flatten -> Dense(128, relu) -> Dense(1, sigmoid)`

Motivación

- Aumentar capacidad para modelar variaciones inducidas por augmentación.

Experimento B4 — Resultados

- (con aug, sin capa oculta) **Test**: **65.38%**, **F1**: **0.6301**
- (con aug + Dense(128,relu)) **Train**: **82.04%**
- (con aug + Dense(128,relu)) **Test**: **78.21%**, **F1**: **0.8468**
- La capa oculta recupera gran parte del rendimiento perdido por augmentación.

Comparación Global

Tabla resumen — Flores (test accuracy)

Método	Arquitectura (alto nivel)	Test Acc
Manual OVR + sigmoid	K*logistic (lineal)	**35.71%**
Manual Softmax	Softmax lineal	**35.71%**
Keras baseline	Dense(K, softmax)	**42.86%**
Keras + aug	Flatten + Dense(K, softmax)	**45.24%**
Keras + aug + hidden	Flatten + Dense(128,relu) + Dense(K)	**28.57%**

Tabla resumen — Pokémon (test)

Método	Arquitectura (alto nivel)	Test Acc	F1
Manual sigmoid	Logistic (lineal)	**91.03%**	**0.9346**
Keras baseline	Dense(1, sigmoid)	**83.33%**	**0.8738**
Keras + aug	Flatten + Dense(1, sigmoid)	**65.38%**	**0.6301**
Keras + aug + hidden	Flatten + Dense(128,relu) + Dense(1)	**78.21%**	**0.8468**

Conclusiones

Conclusiones (1/2)

- En **Flores**, los modelos lineales (manuales o Keras) tienden a:
  - Ajustar perfecto en train (100%)
  - Generalizar poco en test (~36–45%)
- La **matriz de confusión** confirma que el modelo no separa bien clases similares.

Conclusiones (2/2)

- La **augmentación** es contextodependiente:
  - Puede ayudar ligeramente (Flores: 42.86% → 45.24%)
  - Puede perjudicar si el modelo no tiene capacidad/tuning (Pokémon lineal: 83.33% → 65.38%)
- Agregar **no-linealidad** (Dense+ReLU) ayuda en Pokémon con aug, pero en Flores no fue suficiente.