Tecnológico de Monterrey
Escuela de Ingeniería y Ciencias

Clasificación de Balones Deportivos Utilizando Redes Neuronales Convolucionales (CNN)

Autor: Frida Bailleres González

Abstract

En este trabajo se desarrolla un modelo de red neuronal convolucional (CNN) para la clasificación multiclase de imágenes de balones deportivos. La arquitectura del modelo se inspira en el enfoque ICNN-BNDOA propuesto por Ogundokun et al. (2022), adaptando Dropout como técnica de regularización. Para la función de pérdida *(loss)* se utilizó *categorical cross-entropy* y como métrica principal se seleccionó *accuracy*, respaldada por el análisis de Terven et al. (2025) sobre métricas en aprendizaje profundo. El modelo final alcanzó una *accuracy* del 87.5% en el conjunto de prueba. Los resultados se validan mediante gráficas de entrenamiento y una matriz de confusión.

Key Words

Clasificación de imágenes, Red neuronal convolucional, ICNN-BNDOA, Métricas, *Categorical cross-entropy*. *Accuracy*, Balones deportivos, Reconocimiento de patrones, Dropout, Matriz de Confusión.

i. Introducción

El reconocimiento de imágenes es una de las aplicaciones en la actualidad con más impacto en campos como la medicina, el transporte, la seguridad y el deporte. La clasificación de objetos visuales en imágenes se ha beneficiado significativamente del uso de redes neuronales convolucionales (CNN) las cuales han demostrado ser altamente eficaces para detectar patrones complejos.

Este trabajo se enfoca en la tarea de clasificación multiclase de imágenes de balones deportivos, esto implica diferenciar visualmente entre objetos con formas similares, pero con texturas, colores y patrones distintos.

La arquitectura del modelo se inspira en ICNN-BNDOA, una CNN mejorada basada en el uso de *Batch Normalization*, *Dropout* y el optimizador Adam (Ogundokun et al., 2022).

La evaluación del modelo se realizó mediante métricas de clasificación multiclase, con énfasis en *accuracy*, seleccionada por su efectividad en contextos balanceados según lo mencionado por Terven et al. (2025).

ii. Repaso literario

Este proyecto se fundamenta en dos trabajos recientes del estado del arte que abordan los dos componentes clave de un modelo de clasificación basado en redes neuronales: la arquitectura de la red y la selección de métricas adecuadas. En primer lugar, el artículo de Ogundokun et al. (2022) propone una arquitectura denominada ICNN-BNDOA, que combina el uso de capas convolucionales con técnicas de regularización como *Batch Normalization* y *Dropout*, y emplea el optimizador Adam. Esta arquitectura mostró mejoras significativas en la clasificación binaria de imágenes, sirviendo como punto de partida para adaptar un modelo multiclase ajustado a un conjunto de datos más pequeño.

En segundo lugar, el artículo de Terven et al. (2025) ofrece un análisis exhaustivo de funciones de *loss* y métricas en aprendizaje profundo. En su revisión, los autores destacan que la métrica de accuracy sigue siendo la más utilizada y adecuada en tareas de clasificación multiclase cuando los conjuntos de datos están balanceados y no existen penalizaciones diferenciadas por clase. También reafirman la utilidad de *categorical cross-entropy* cuando se utiliza una capa de salida con *softmax*, como en el modelo presentado. Ambos artículos contribuyeron a orientar tanto el diseño del modelo como la estrategia de evaluación.

iii. Dataset y Preprocesado

Se utilizó el conjunto de datos “Sports Balls – Multiclass Image Classification” disponible en Kaggle. Aunque el dataset original contenía múltiples categorías, se seleccionaron únicamente cuatro clases relevantes para el objetivo del modelo: cricket, football, soccer y tennis.

Durante la exploración inicial, se detectaron algunas imágenes que no eran representativas. Por tanto fue necesario realizar una limpieza manual del dataset, eliminando las imágenes que no reflejaban correctamente su categoría o que presentaban baja calidad visual. Igualmente, se observó que el número de imágenes por clase era insuficiente para entrenar una red convolucional de forma efectiva, por lo que se implementó un script en python para extraer imágenes aleatorias de internet y utilizarlas en el dataset. Posteriormente, se volvieron a revisar las imágenes seleccionadas por el script con la finalidad de asegurar que hayan sido correctas y no inyectara ruido en el dataset que pudiera ser contraproducente.

El dataset final fue dividido en tres subconjuntos: *train* (80%), *validation (val)* (10%) y *test* (10%), asegurando un reparto equilibrado entre clases. Cada imagen fue redimensionada a 150x150 píxeles y normalizada mediante escalamiento a valores entre [0, 1].

Tamaño de train	***Tamaño de test y val***	***Número de clases***
2,800	280 cada una	4

Tabla 1. Cantidad de imágenes por subconjunto y número de clases

Figura 1. Distribución de dataset en los 3 subconjuntos

Para el entrenamiento se aplicaron técnicas de preprocesamiento con *ImageDataGenerator*, incluyendo rotación, desplazamientos horizontales y verticales, zoom, cambios de brillo y recortes, con el objetivo de aumentar la variabilidad del conjunto y mejorar la generalización del modelo. Los subconjuntos de *val* y *test* fueron únicamente reescalados, sin alteraciones.

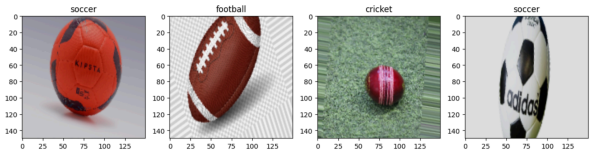

Figura 2. Imágenes ya preprocesadas del subconjunto *train*.

iv. Implementación

La arquitectura implementada en este proyecto es una red neuronal convolucional (CNN) inspirada en la estructura ICNN-BNDOA, pero adaptada a un conjunto de datos pequeño y balanceado. A diferencia del modelo original, que combina **Batch *Normalization*** *(BN)* con *Dropout (DO)*, se decidió eliminar *BN* debido a que durante la experimentación, esta técnica disminuyó la precisión del modelo. Esto se alinea con hallazgos que sugieren que *BN* puede ser contraproducente en datasets reducidos, donde la estimación de la media y varianza por batch no es confiable.

"Batch normalization works best with large datasets and batch sizes, and its effectiveness can be limited when data is scarce or imbalanced." (Goodfellow et al., Deep Learning, 2016)

El modelo incluye tres bloques convolucionales. Cada bloque está compuesto por una capa **Conv2D** (con 16, 32 y 64 filtros respectivamente, todos de tamaño 3x3), que se encarga de extraer características visuales como bordes, texturas o formas en diferentes niveles de profundidad. Estas capas utilizan la activación **ReLU** (Rectified Linear Unit), que mantiene los valores positivos y descartar los negativos, lo cual también acelera el entrenamiento. Después de cada capa convolucional se agrega una capa de **MaxPooling2D** (de un tamaño de 2x2), que reduce el tamaño de la imagen conservando la información más importante, esto ayuda a disminuir el tiempo de entrenamiento y a evitar el *overfitting*. A partir del segundo bloque, se incorporan capas de **Dropout(0.2)**, que desactivan aleatoriamente ciertas neuronas durante el entrenamiento para que el modelo no dependa demasiado de algunos patrones y generalice mejor. En el tercer bloque se utiliza una capa **Flatten**, que convierte la salida tridimensional de las capas anteriores en un vector unidimensional, el cual es necesario para conectarlo con las capas densas (fully connected). Luego se aplica una tercera capa de **Dropout(0.3)** como paso final antes de la etapa de clasificación.

Las capas finales del modelo son un **Dense(64)** con activación **ReLu**, seguido de una capa **Dense** con 4 neuronas, correspondiente al número de clases: cricket, football, soccer y tennis. Esta capa utiliza activación **softmax**, esta activación asegura que las salidas sumen 1, lo cual es ideal para problemas de clasificación de múltiples clases.

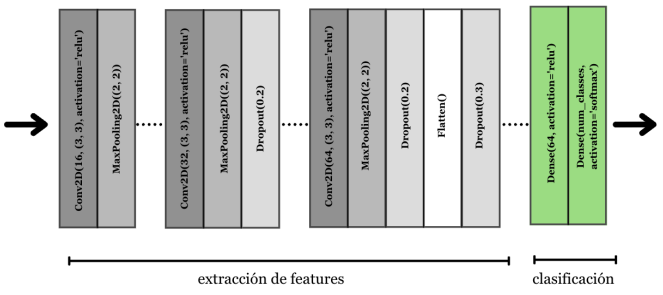

Figura 3. Representación de la arquitectura utilizada

iiv. Métricas Utilizadas

El modelo fue compilado con la función de *loss* **categorical cross-entropy**, definida como:

$$CE = -\sum_{i=1}^{C} t_i \cdot \log(f(s)_i)$$

Donde C es el número de clases, t_i representa el valor verdadero para la clase i ("1" es clase correcta y "0" si no lo es), y $f(s)_i$ es la probabilidad predicha por el modelo para esa clase i. Esta función penaliza con mayor severidad cuando el modelo se equivoca con alta confianza, lo cual ayuda a mejorar las predicciones.

Como optimizador se utilizó **Adam** (Adaptive Moment Estimation), el cual ajusta automáticamente la tasa de aprendizaje durante el entrenamiento. Se seleccionó un *learning_rate=1e-4*, ya que es el valor que recomienda el estado del arte que se seleccionó como base.

El modelo fue entrenado durante **50 épocas** con un tamaño de batch de 8 para el *train*. El conjunto de *val* se utilizó para monitorear el rendimiento y prevenir el *overfitting*. Al finalizar el entrenamiento, se evaluó el modelo con el conjunto de *test*, calculando métricas como *accuracy*, matriz de confusión y reporte de clasificación por clase con más métricas.

v. Resultados

Tras entrenar el modelo durante 50 épocas, se obtuvo un *accuracy* final de **87.5%** en el conjunto de prueba, con un *loss* de **0.4036**, lo cual indica un buen nivel de generalización sin sobreajuste significativo. Las curvas de entrenamiento muestran una mejora progresiva tanto en accuracy como en *loss*, y una convergencia estable entre los subconjuntos de entrenamiento y validación. Esto sugiere que el modelo fue capaz de aprender patrones relevantes sin memorizar los datos.

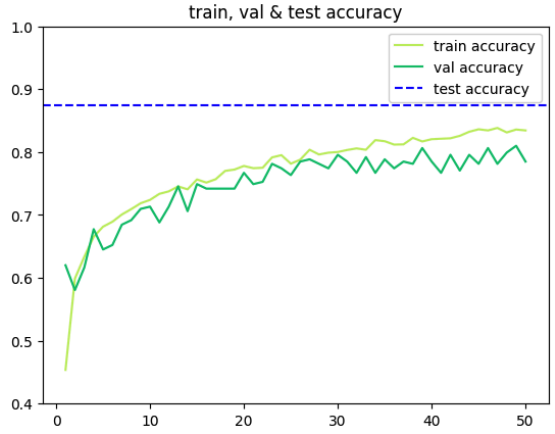

Figura 4. *Accuracy* durante las 50 épocas.

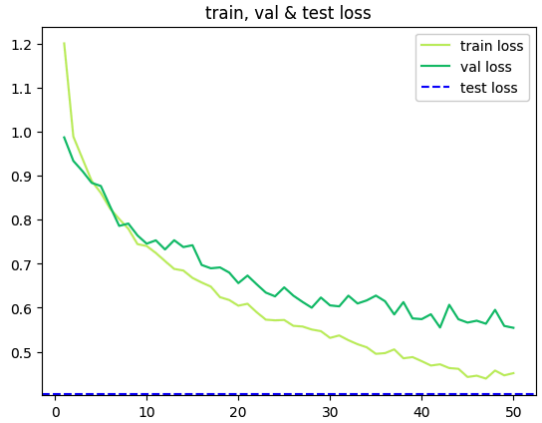

Figura 5. *Loss* durante las 50 épocas.

En cuanto a las métricas por clase, el modelo mostró un desempeño sólido y balanceado. Para entender los resultados, es importante distinguir entre las métricas utilizadas. **Accuracy** (o exactitud) indica el porcentaje total de predicciones correctas del modelo sobre el total de imágenes, y en este caso fue de **87.5%**, lo que significa que acertó en la mayoría de los casos. **Precision** mide qué tan bien el modelo acierta cuando predice una clase específica, es decir, de todas las veces que dijo "es football", cuántas veces realmente lo fue. Por ejemplo, la clase **football** tuvo una **precisión de 0.98**, lo que muestra que el modelo casi no se equivocó cuando

dijo que una imagen era de football. **Recall**, por otro lado, mide cuántos casos reales de una clase fueron detectados correctamente; por ejemplo, **cricket** tuvo un **recall de 0.93**, lo que quiere decir que el modelo identificó correctamente el 93% de las imágenes de cricket reales. El **F1-score**, que combina *precision* y *recall* en una sola métrica, fue mayor a 0.83 en todas las clases. Además los promedios fueron de **0.875**, lo que indica que el modelo mantuvo un rendimiento estable entre todas las clases.

	precision	recall	f1-score	support
cricket	0.76	0.93	0.83	70
football	0.98	0.86	0.92	70
soccer	0.93	0.81	0.87	70
tennis	0.88	0.90	0.89	70
accuracy			0.88	280
macro avg	0.89	0.87	0.88	280
weighted avg	0.89	0.88	0.88	280

Figura 6. Reporte de clasificación del modelo.

La matriz de confusión complementa este análisis visualizando el desempeño del modelo en cada clase. Se observa que cricket y tennis fueron las clases mejor clasificadas, con 65 y 63 aciertos respectivamente. Football mostró una pequeña confusión con cricket (7 casos), y soccer fue confundida ocasionalmente con cricket y tennis. Sin embargo, el nivel general de aciertos fue alto en todos los casos.

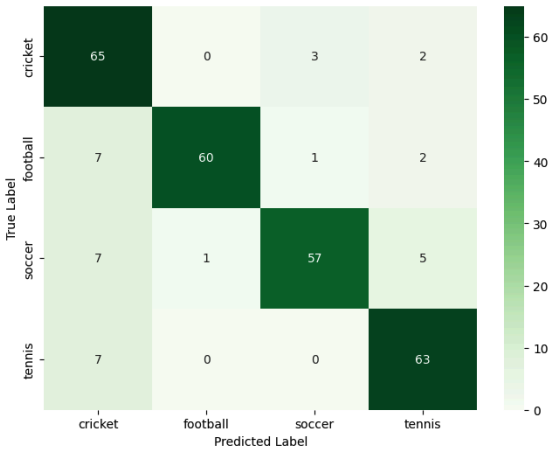

Figura 7. Matriz de confusión.

Estos resultados validan que la arquitectura propuesta, junto con las decisiones tomadas en cuanto a regularización y preprocesamiento, fue efectiva para resolver la tarea de clasificación multiclase con un dataset balanceado pero limitado en tamaño.

vi. Comparación Contra otro Algoritmo

Antes de llegar a la arquitectura final, se probó una red más profunda y compleja con tres capas **Conv2D** de 32, 64 y 128 filtros respectivamente, intercaladas con capas de **Bach Normalization**, y seguida por una capa **Dense(256)** antes de la salida softmax. Aunque esta arquitectura parecía más robusta, sus resultados fueron inferiores: obtuvo un *accuracy* de tan solo **47.6%** en el conjunto de *test* y una pérdida *(loss)* muy elevada de 83.8.

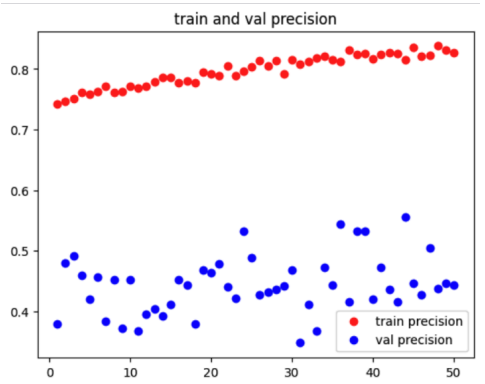

Figura 8. Precisión durante las 50 épocas de la primera arquitectura probada.

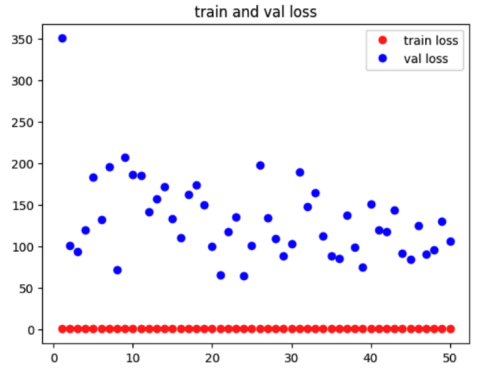

Figura 9. *Loss* durante las 50 épocas de la primera arquitectura probada.

Una de las causas principales fue que esta red era demasiado compleja para el tamaño del dataset original (antes de que fuera ajustado a como se muestra en la Tabla 1):

Tamaño de train	*Tamaño de test y val*	*Número de clases*
2,000	200 cada una	4

Tabla 2. Cantidad de imágenes por subconjunto y número de clases de primer arquitectura

Esto causó *overfitting*. Es importante mencionar que las imágenes de *val* y *test* no estaban correctamente escaladas al rango [0, 1], lo que afectó drásticamente la pérdida durante la evaluación.

```
Test loss: 83.80927276611328
Test accuracy: 0.47600001096725464
Test precision: 0.48373982310295105
```

Figura 10. Resultados de *loss*, *accuracy* y precisión de la primera arquitectura probada.

Arquitectura	Test Accuracy	Test Precision	Test Loss
Primera	0.476	0.483	83.81
Final	0.875	0.89	0.4036

Tabla 3. Comparación de resultados de ambas arquitecturas.

También se detectó que muchas imágenes del dataset original no eran representativas de su clase, lo cual generaba ruido durante el entrenamiento. Todo esto llevó a reconsiderar la estrategia y optar por una arquitectura más simple, al igual que limpiar y aumentar el dataset utilizado.

Con la arquitectura final (basada en ICNN-BNDOA), el modelo alcanzó un **87.5%** de *accuracy* y una pérdida *(loss)* de **0.4036**, lo que representa una mejora significativa tanto en rendimiento como en estabilidad.

vii. Conclusiones

Este proyecto me permitió aplicar de forma práctica los conceptos de este bloque "Desarrollo de aplicaciones avanzadas de ciencias computacionales", como la generación y depuración de un dataset, la selección e implementación de arquitecturas respaldadas por el estado del arte, y el uso de métricas adecuadas para la evaluación de modelos. También entendí la importancia del preprocesamiento y el ajuste de hiperparámetros para evitar problemas como el *overfitting*. Más allá de lograr un buen desempeño, comprobé que un modelo exitoso no solo depende de su arquitectura, sino de un flujo completo bien diseñado, como se discutió a lo largo de esta clase.

viii. Referencias

[1] R. O. Ogundokun, R. Maskeliunas, S. Misra y R. Damaševičius, "Improved CNN Based on Batch Normalization and Adam Optimizer," en Computational Science and Its Applications – ICCSA 2022 Workshops, Lecture Notes in Computer Science, vol. 13381, O. Gervasi et al., Eds. Cham: Springer, 2022, pp. 384–394. doi: 10.1007/978-3-031-10548-7_43

[2] J. Terven, D. M. Cordova-Esparza, J. A. Romero-González, et al., "A comprehensive survey of loss functions and metrics in deep learning," Artificial Intelligence Review, vol. 58, p. 195, 2025. doi: 10.1007/s10462-025-11198-7

[3] I. Goodfellow, Y. Bengio y A. Courville, Deep Learning, MIT Press, 2016. [Online]. https://www.deeplearningbook.org