PCD - Análisis de Imágenes

Ciencia de Datos e IA - 4º
Javier Pérez Vargas
Mario Ruiz Vaquett

Diciembre 2024

1. Introducción

En este proyecto, realizaremos un análisis comparativo de diferentes arquitecturas de redes neuronales convolucionales (CNNs) aplicadas a la tarea de clasificación de imágenes. Estas arquitecturas han demostrado ser herramientas fundamentales en el campo de la visión por computadora debido a su capacidad para extraer y aprender características jerárquicas de los datos visuales. Nuestro objetivo será evaluar el desempeño de varias CNNs utilizando un conjunto de datos sobre imágenes de diferentes deportes.

2. Conjunto de Datos

El conjunto de datos utilizado en este proyecto es el siguiente: Sports Classification, diseñado para tareas de clasificación de deportes. Este dataset contiene imágenes etiquetadas de una amplia variedad de deportes. Las imágenes abarcan tanto deportes individuales como de equipo, así como actividades realizadas en diversos entornos.

Debido a la amplia cantidad de categorías disponibles, hemos decidido limitar nuestro análisis a 10 deportes específicos para simplificar el alcance del proyecto. Las categorías seleccionadas son: *basketball*, *golf*, *hockey*, *formula 1 racing*, *football*, *bowling*, *surfing*, *sumo wrestling*, *tennis* e *ice climbing*. Esta selección incluye una mezcla de deportes populares y actividades menos comunes, asegurando una variedad interesante de contextos visuales. A continuación podemos ver una pequeña muestra:

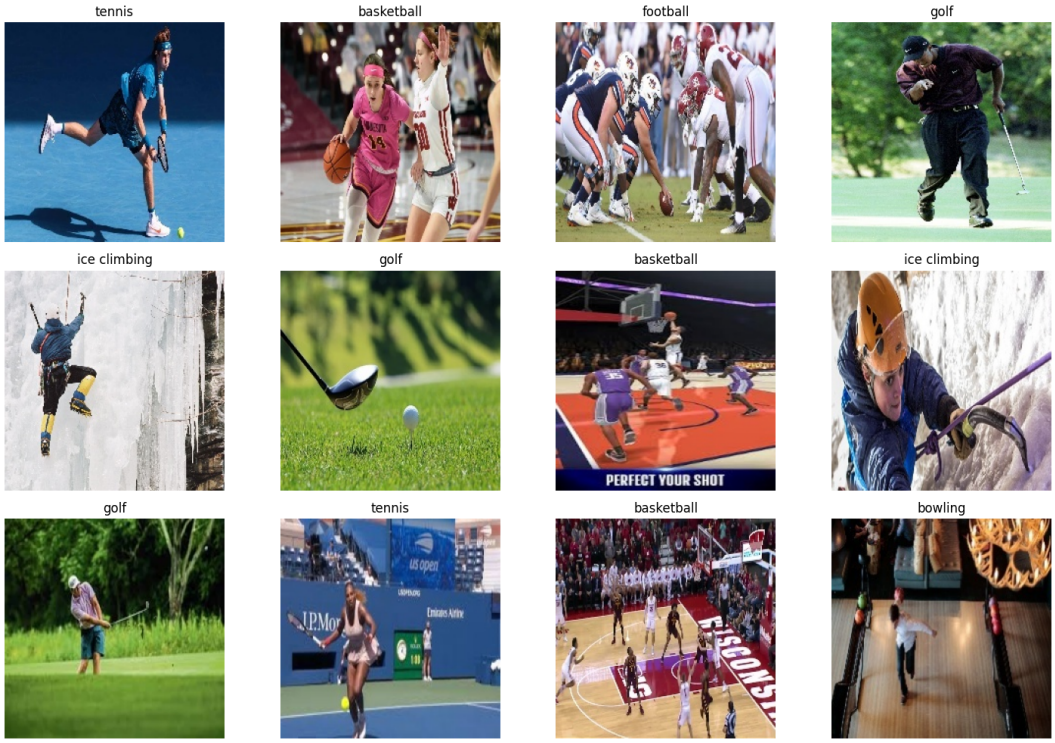

Figura 1: Muestra de las imágenes del dataset

En total, tenemos 1535 imágenes de entrenamiento, con unas 140-150 de cada clase, aproximadamente, de manera balanceada. En cuanto al conjunto de validación y conjunto de test, tenemos 5 imágenes en cada grupo por cada una de las 10 clases.

3. Metodología

Para realizar el proyecto seguiremos un proceso que constará de varias etapas:

1. Diseñar una **clase personalizada para gestionar el conjunto de datos**, incluyendo técnicas de *data augmentation* para enriquecer los datos de entrenamiento y mejorar la generalización.
2. Implementar cada **arquitectura de CNN** especificada, destacando sus diferencias clave en términos de diseño y funcionamiento.
3. **Evaluar y comparar** el desempeño de las CNNs en el conjunto de datos de validación, utilizando métricas como precisión y pérdida.
4. Realizar un **estudio de ablación** para analizar el impacto de los componentes individuales de las arquitecturas sobre su rendimiento global.

3.1. Dataset Personalizado

El dataset está dividido en tres subconjuntos: entrenamiento, validación y prueba. Las imágenes del subconjunto de entrenamiento fueron procesadas mediante dos tipos de transformaciones:

- **Transformación Original:** Incluye la conversión a tensores (conveniente cuando usamos PyTorch) y la normalización utilizando los valores medios y desviaciones estándar adecuados para la entrada del modelo. Estos valores de normalización son típicamente utilizados en este tipo de modelos.
- **Transformación Aumentada:** Además de las operaciones de la transformación original, incluye técnicas de aumento de datos como voltear horizontalmente las imágenes, rotaciones aleatorias en un rango de $[-30°, 30°]$ y ajustes aleatorios en brillo, contraste, saturación y matiz. Estas transformaciones aumentan la diversidad de los datos de entrenamiento, ayudando al modelo a generalizar mejor.

El subconjunto de validación y prueba empleó únicamente la transformación original, garantizando que estas imágenes no sean alteradas por aumentos y representen de manera consistente la calidad de los datos en el entorno real.

Para gestionar de manera eficiente los datos y aprovechar la flexibilidad ofrecida por PyTorch, hemos creado una clase personalizada que hereda de `Dataset`. Esta clase nos permite integrar fácilmente las transformaciones y gestionar los datos de manera modular. Además, esto permitirá cargar los datos utilizando *DataLoaders*.

```python
class CustomImageDataset(Dataset):
    def __init__(self, directory, transform=None):
        self.directory = directory
        self.transform = transform
        self.image_paths = []
        self.labels = []

        # Obtener las clases (subdirectorios) y asignar un número
        self.class_names = sorted(os.listdir(directory))
        self.class_to_idx = {class_name: idx for idx, class_name in enumerate(self.class_names)}

        # Recorremos las carpetas y archivos
        for class_name in self.class_names:
            class_path = os.path.join(directory, class_name)
            if os.path.isdir(class_path):
                for file_name in os.listdir(class_path):
                    if file_name.endswith(('.jpg', '.png')):  # Filtra los tipos de archivo
                        self.image_paths.append(os.path.join(class_path, file_name))
                        self.labels.append(self.class_to_idx[class_name])

    def __len__(self):
        return len(self.image_paths)
```

```python
def __getitem__(self, idx):
    img_path = self.image_paths[idx]
    label = self.labels[idx]
    img = Image.open(img_path)

    if self.transform:
        img = self.transform(img)

    return img, label
```

3.2. Modelos

En esta sección se describen los diferentes modelos de aprendizaje profundo empleados para la clasificación de las imágenes del conjunto de datos. Se exploraron tres enfoques principales: una red neuronal convolucional (CNN) sencilla para establecer una línea base, una CNN residual con una arquitectura avanzada, y un modelo basado en *Transfer Learning*, que aprovecha redes preentrenadas en grandes conjuntos de datos.

Los hiperparámetros utilizados en las redes son los siguientes:

- **Loss function** (`CrossEntropyLoss`): Se usa para problemas de clasificación multiclase, ya que calcula la diferencia entre las probabilidades predichas y las etiquetas reales.
- **Learning rate** (`1e-3`): Se han probado varios valores pero este permite una convergencia estable sin grandes saltos durante el entrenamiento.
- **Optimizer** (`Adam`): Se selecciona debido a su capacidad para adaptarse automáticamente a los gradientes, mejorando la eficiencia en la optimización.
- **Number of epochs** (`50`): Hemos elegido este número tras varios intentos, con el objetivo de que el modelo tenga tiempo de aprender patrones sin sobreajustarse a los datos.

3.2.1. Red Convolucional Simple

Esta arquitectura es una red convolucional sencilla diseñada para tareas de clasificación de imágenes. Está compuesta por varias capas convolucionales seguidas de capas densas para la clasificación final.

Arquitectura General

La arquitectura del modelo *CNN* se estructura de la siguiente manera:

1. **Capas Iniciales**: Comienza con tres capas convolucionales, cada una seguida de una operación de *ReLU* y *Max Pooling* para extraer características progresivamente más complejas y reducir la resolución espacial de las imágenes.
2. **Capas Completamente Conectadas**: Después de las capas convolucionales, la salida se aplana y pasa a través de dos capas densas: la primera con 512 neuronas y la segunda con 10 neuronas, que corresponden a las clases del conjunto de datos.

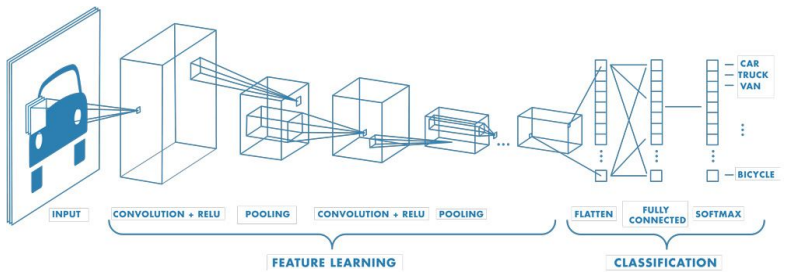

Figura 2: Ejemplo de CNN clásica

Esta arquitectura simple es adecuada para modelos de clasificación de imágenes que no requieren una gran profundidad, pero que aún pueden aprender representaciones efectivas de las imágenes para tareas de clasificación.

3.2.2. Red Convolucional Residual

Esta arquitectura se basa en los principios de las Redes Residuales (*ResNet*), que utilizan bloques residuales para mitigar el problema del desvanecimiento del gradiente y mejorar el entrenamiento en redes profundas. Los ***ResidualBlock*** implementan una conexión residual directa entre la entrada y la salida del bloque. Cada bloque residual consta de las siguientes operaciones:

- Dos capas convolucionales con un tamaño de kernel de 3×3, seguidas de capas de normalización por lotes (*Batch Normalization*) y funciones de activación *ReLU*.
- Una conexión directa (*shortcut*) que ajusta la dimensionalidad de la entrada si es necesario mediante una convolución 1×1 (cuando las dimensiones de entrada y salida difieren) o la deja sin cambios (*Identity*).
- La salida del bloque es la suma de la conexión directa y la salida de las capas convolucionales, seguida por la función de activación *ReLU*.

El diseño de este bloque permite preservar información a lo largo de las capas de la red, lo que resulta en un mejor desempeño en tareas complejas.

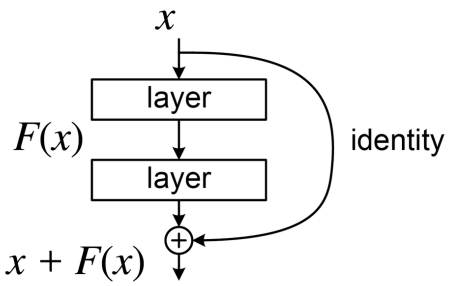

Figura 3: Funcionamiento básico de un bloque residual

Arquitectura General

La arquitectura general de este segundo modelo es la siguiente:

1. **Capas Iniciales**: Una capa convolucional inicial con 32 canales y un tamaño de kernel de 3×3, seguida de una capa de normalización por lotes y una operación de *Max Pooling* para reducir la resolución espacial.
2. **Bloques Residuales**: Dos bloques residuales consecutivos que aumentan progresivamente el número de canales de salida (de 32 a 64, y de 64 a 128) mientras reducen la resolución espacial mediante convoluciones con un stride de 2.
3. **Capas Completamente Conectadas**: La salida de los bloques residuales es aplanada y pasada a través de una serie de capas densas:
   - Una capa lineal dinámica que ajusta su tamaño a la dimensión de la salida de los bloques residuales, seguida por una capa de 256 neuronas con activación *ReLU* y *Dropout*.
   - Una capa de 128 neuronas, también con activación *ReLU* y *Dropout*.
   - Una capa de salida con 10 neuronas, lo que corresponde a las categorías que hemos seleccionado, y activación lineal.

Se espera que esta segunda red ofrezca un rendimiento superior al de la primera, logrando una mayor precisión, aunque a costa de un mayor tiempo de entrenamiento.

3.2.3. Red Basada en Transfer Learning

El **Transfer Learning** aprovecha características extraídas por modelos preentrenados en grandes conjuntos de datos, como *ImageNet*, y permite adaptarlas a tareas específicas con modificaciones mínimas. Esto es ideal para problemas con datos limitados, como en nuestro caso, ya que evita entrenar un modelo desde cero.

El modelo base utilizado es **DenseNet121**, una red profunda conocida por su diseño eficiente, que conecta densamente cada capa con todas las anteriores, maximizando la reutilización de características. Las capas convolucionales de DenseNet (bloque *features*) se mantienen congeladas para preservar sus pesos preentrenados. Además, se ha realizado una pequeña modificación en el clasificador de DenseNet para reducir el overfitting y especificar el número de clase, y será especificada a continuación.

Arquitectura General

La arquitectura incluye los siguientes cambios principales:

1. **Extracción de Características:** Se utiliza el bloque *features* de DenseNet121 para obtener representaciones profundas.
2. **Clasificador Personalizado:** La capa final original, se reemplaza por:
   - Una capa densa con 512 neuronas y activación $ReLU$.
   - Regularización con *Dropout* (50 %).
   - Una capa final que produce la salida para las 10 clases objetivo.

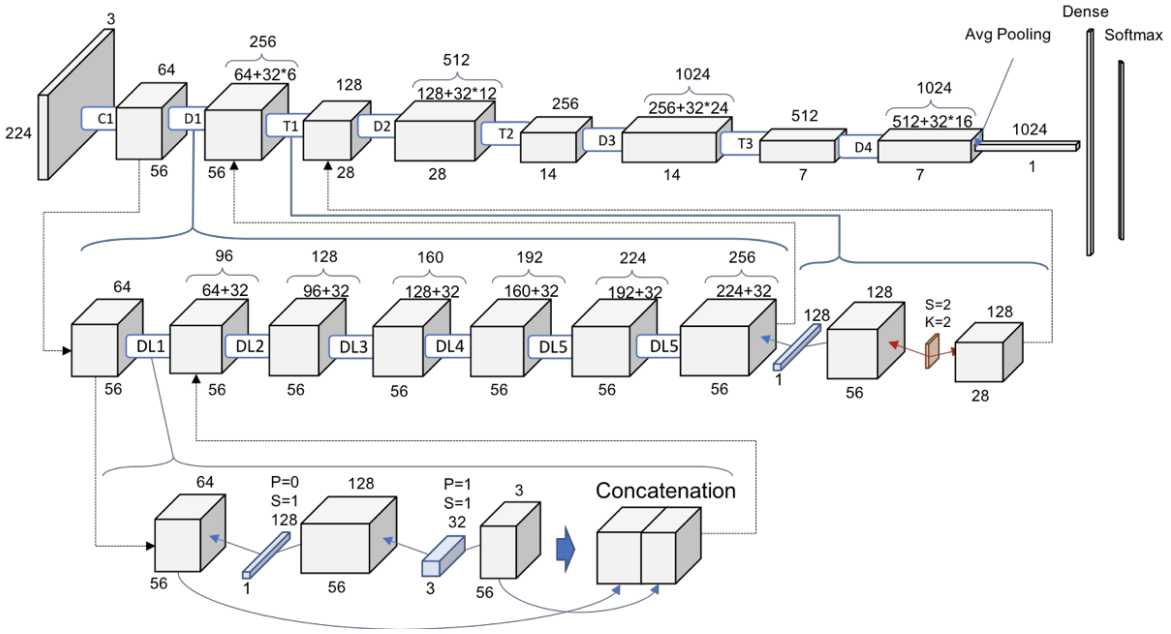

Figura 4: Arquitectura de la red DenseNet

Esta configuración combina la potencia del modelo preentrenado con un clasificador adaptado, optimizando tanto la generalización como el ajuste al problema.

3.3. Resultados

	Modelo	Datos	Epochs	Train Acc.	Val Acc.
Experimento 1	CNN Básica	Original	50	100 %	72 %
Experimento 2	CNN Básica	Augmented	50	97 %	86 %
Experimento 3	CNN Residual	Original	50	97 %	90 %
Experimento 4	CNN Residual	Augmented	50	78 %	96 %
Experimento 5	Transfer Learning	Original	50	92.9 %	100 %
Experimento 6	Transfer Learning	Augmented	50	83.7 %	98 %

Cuadro 1: Resultados de los experimentos con diferentes modelos y configuraciones.

3.4. Resultados en el conjunto de Test

Hemos decidido escoger el modelo obtenido del experimento 4 para evaluar en el conjunto de test, pues a pesar de tener una precisión mejorable en el conjunto de entrenamiento, ha demostrado ser uno de los mejores modelos a la hora de generalizar a nuevos datos.

Así, en el conjunto de test se ha obtenido un **86 %** de accuracy. Es menos de lo esperado tras el 96 % en validación, pero debemos recordar que los conjuntos de validación y test eran muy pequeños (5 fotos en cada conjunto por cada una de las 10 clases), y es posible que la variabilidad de los conjuntos haya influido. A pesar de ello, un 86 % es un resultado aceptable y refleja que el modelo tiene un desempeño adecuado para el problema planteado.

3.5. Estudio de ablación

El estudio de ablación tiene como objetivo analizar el impacto de diferentes configuraciones en el rendimiento de los modelos de clasificación. En la Tabla 1, se presentan los resultados obtenidos en seis experimentos donde se evaluó el efecto de dos factores principales:

- **Tipo de Modelo:** Se comparó el rendimiento de una CNN básica, una CNN con bloques residuales y un modelo basado en aprendizaje por transferencia.
- **Aumentación de Datos:** Se incluyó una versión 'augmented' del conjunto de datos original para observar su impacto en la capacidad de generalización de los modelos.

En la primera de las redes, observamos un posible overfitting cuando utilizamos los datos originales, dado que la precisión en el conjunto de entrenamiento es notablemente mayor que en el conjunto de validación. Esto se puede deber a que no tenemos ninguna capa relativa a la regularización (L1, L2, Dropout...) en nuestra CNN básica. No obstante, sí es interesante el aumento en rendimiento que se produce cuando utilizamos los datos aumentados, ya que la red se ajusta mucho mejor a los datos de validación (86 %), superando con creces al experimento anterior, y sin sobreajustarse tanto al conjunto de entrenamiento. En general, la CNN es bastante básica aunque proporciona unos resultados mejores de lo esperado. Eliminar profundidad a la red (capas o neuronas) supondría perder capacidad de aprendizaje, aunque también permitiría un menor sobreentrenamiento.

Por otra parte, la CNN residual muestra una dinámica similar, aunque con porcentajes de precisión en validación superiores a los experimentos anteriores. En este caso, las diferencias eran los bloques residuales, que han permitido utilizar una red más profunda sin peligro del desvanecimiento de gradiente. Además, en este caso sí existían componentes de regularización (Dropout), que han podido influir en la buena generalización del modelo. Vemos que en el conjunto de entrenamiento se ha sobreajustado menos que los experimentos anteriores, especialmente con el dataset aumentado, donde no ha conseguido captar todas las diferencias en el conjunto de entrenamiento pero ha servido para poder distinguir las clases en validación, con un alto porcentaje de precisión (96 %). A continuación vemos una imagen del entrenamiento:

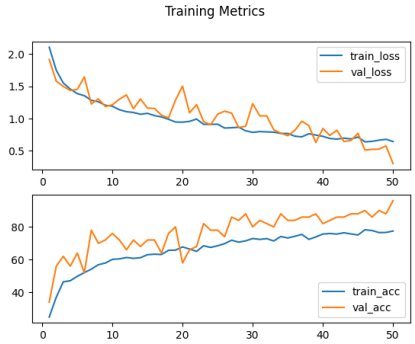

Figura 5: Accuracy y loss a lo largo de las épocas con datos aumentados en el modelo residual

Este es el entrenamiento para el modelo que hemos elegido finalmente. Observamos que el aprendizaje aumenta a lo largo de las épocas, y que incluso alguna época más hubiera sido beneficioso para el modelo. Además, observamos cómo el conjunto de entrenamiento siempre está por debajo (en accuracy) que el conjunto de validación, debido a que se está entrenando con el conjunto aumentado y es incapaz de sobreajustarse.

Finalmente llegamos a los experimentos con Transfer Learning. Parece incoherente pero en ambos casos se ha obtenido una precisión mayor en el conjunto de validación que en el conjunto de entrenamiento. Esto lo asociamos a dos causas: O bien al posible exceso de regularización con las capa dropout añadida, o bien a la simplicidad del conjunto de validación. Nos inclinamos un poco más por la última razón, pues es un conjunto muy pequeño, donde no hay deportes que se solapen mucho en las imágenes, y que permite un aprendizaje rápido. Como vemos, de nuevo, en el conjunto aumentado la precisión en entrenamiento es menor. No obstante, en ambos experimentos se ha obtenido una precisión en test casi perfecta.

4. Conclusiones

En este proyecto hemos comparado diferentes arquitecturas de redes neuronales para la tarea de clasificación de imágenes de deportes, analizando cómo las técnicas empleadas afectan el rendimiento de los modelos. Las redes más simples, como la CNN básica, presentaron limitaciones en su capacidad de generalización, especialmente al trabajar con datos sin aumentación. En contraste, la CNN residual mostró un mejor equilibrio entre precisión y generalización gracias a la implementación de bloques residuales y técnicas de regularización. Por último, el uso de Transfer Learning con DenseNet121 se destacó como una solución muy efectiva, logrando resultados excepcionales en las fases de validación.

La aumentación de datos ha sido un factor crucial para mejorar el rendimiento de los modelos, ayudando a mitigar el sobreajuste y a potenciar la capacidad de generalización. En particular, las redes entrenadas con datos aumentados mostraron un rendimiento superior en validación, resaltando la relevancia de esta técnica.

Cabe destacar el pequeño tamaño del dataset, pues no solo limita la cantidad de información disponible para que el modelo aprenda patrones, sino que también incrementa la probabilidad de que los conjuntos de validación y test no representen adecuadamente la distribución real de las clases. Esto puede generar variaciones significativas en las métricas finales.

En conclusión, PyTorch ha demostrado ser una herramienta muy flexible, ideal para la creación y ajuste de modelos en tareas complejas como la clasificación de imágenes. Su capacidad para gestionar datasets, personalizar arquitecturas y aplicar técnicas avanzadas, como Transfer Learning, facilita la obtención de resultados óptimos.