Tecnológico de Monterrey

Campus Querétaro, Querétaro

Portafolio Implementación

Inteligencia artificial avanzada para la ciencia de datos II | TC3007

Módulo 2 Implementación de un modelo de deep learning.

Por:

Cristian Chávez Guía - A01710680

01/12/2025

INTRODUCCIÓN

En la actualidad, el avance acelerado de la tecnología ha hecho que la inteligencia artificial forme parte de nuestra vida cotidiana, muchas veces sin que nos demos cuenta. Desde los sistemas de recomendación, los asistentes inteligentes, hasta el reconocimiento facial en los dispositivos móviles, los algoritmos de IA han transformado múltiples tareas que antes dependían completamente de intervención humana.

Dentro de las ramas más relevantes de la IA se encuentra la **visión computacional**, una disciplina que permite a las máquinas interpretar y extraer información útil a partir de imágenes o videos. Gracias a ella, hoy es posible automatizar procesos como la detección temprana de enfermedades mediante imágenes médicas, el reconocimiento facial, el monitoreo de seguridad, la clasificación de objetos, entre muchas otras aplicaciones que aportan eficiencia, velocidad y precisión.

Con el propósito de profundizar en esta área desde un enfoque educativo y práctico, se desarrolló un modelo capaz de **identificar si una persona utiliza lentes o no**, empleando arquitecturas modernas basadas en Deep Learning, específicamente **redes neuronales convolucionales (CNNs)**. Aunque el problema pueda parecer sencillo, su implementación representa una excelente oportunidad para comprender en mayor profundidad cómo operan estas arquitecturas, cómo procesan información visual y qué aspectos influyen en su desempeño.

Además, se implementaron **dos modelos distintos** para abordar el mismo problema, lo que permite analizar comparativamente la influencia de la arquitectura, la complejidad del modelo y el uso de técnicas de Transfer Learning:

- **Una CNN construida manualmente**, diseñada desde cero mediante capas convolucionales, activaciones ReLU, pooling y regularizadores.
- **Un modelo basado en MobileNetV2**, utilizando Transfer Learning para aprovechar representaciones visuales previamente aprendidas por una red entrenada con millones de imágenes.

Ambos modelos fueron entrenados utilizando exactamente el mismo pipeline de datos, el cual incluye normalización, data augmentation y un análisis detallado del balance de clases. La comparación entre sus resultados aporta una perspectiva clara sobre las ventajas y limitaciones de cada enfoque dentro de un caso real de visión computacional.

OBJETIVO DEL PROYECTO

El objetivo principal de este proyecto es entrenar y evaluar modelos de Deep Learning capaces de clasificar rostros en dos categorías: personas con lentes y personas sin lentes.

Concretamente, se busca:

- Construir dos modelos independientes para resolver el mismo problema.
- Entrenarlos bajo un pipeline idéntico para garantizar comparabilidad.
- Evaluar su rendimiento con métricas relevantes de clasificación.
- Determinar cuál arquitectura ofrece un mejor desempeño y por qué.
- Analizar sus errores, sesgos y capacidad de generalización.

DATASET

Para este proyecto se utilizó un dataset disponible en Kaggle llamado:

“FaceCropped — Glasses vs No Glasses Dataset”, publicado por **Shahriyar Mammadli**.

El link es el siguiente:
https://www.kaggle.com/datasets/sehriyarmemmedli/facecropped-glasses-vs-noglasses-dataset

Descripción general

- Total de imágenes: 93,604
- Clases:
    - with_glasses
    - without_glasses
- Formato: imágenes recortadas del rostro
- Divisiones ya hechas por el autor:
    - train/
    - val/
    - test/
- Esto facilita enormemente el entrenamiento, ya que el dataset se encuentra:
    - Limpio
    - Balanceado
    - Ordenado por clases
    - Estandarizado en tamaño y posición del rostro

test - 3803
test\without_glasses
test\with_glasses

train - 76049
without_glasses
train\with_glasses

val - 15210
val\without_glasses
val\with_glasses

PREPARACIÓN DE LOS DATOS

En este caso, a pesar de que el dataset ya se encuentra dividido en las carpetas de entrenamiento, validación y prueba, fue necesario implementar un pipeline que garantizara la correcta lectura, transformación y distribución de las imágenes, de modo que los modelos entrenados recibieran los datos en un formato estandarizado y listo para ser procesado por redes neuronales convolucionales.

El primer paso consistió en la carga de los datos utilizando la función image_dataset_from_directory de TensorFlow, la cual resulta particularmente útil cuando se trabaja con datasets organizados por carpetas. Este método automatiza tareas como el etiquetado de clases, la creación de lotes, el redimensionamiento de las imágenes y la mezcla aleatoria del conjunto de entrenamiento. Gracias a esto, se logró obtener de manera sencilla tres datasets estructurados: uno para entrenar al modelo, otro para hacer validaciones en tiempo real y un último para evaluar el rendimiento final. Además, se fijó una semilla aleatoria para asegurar que los experimentos fueran reproducibles.

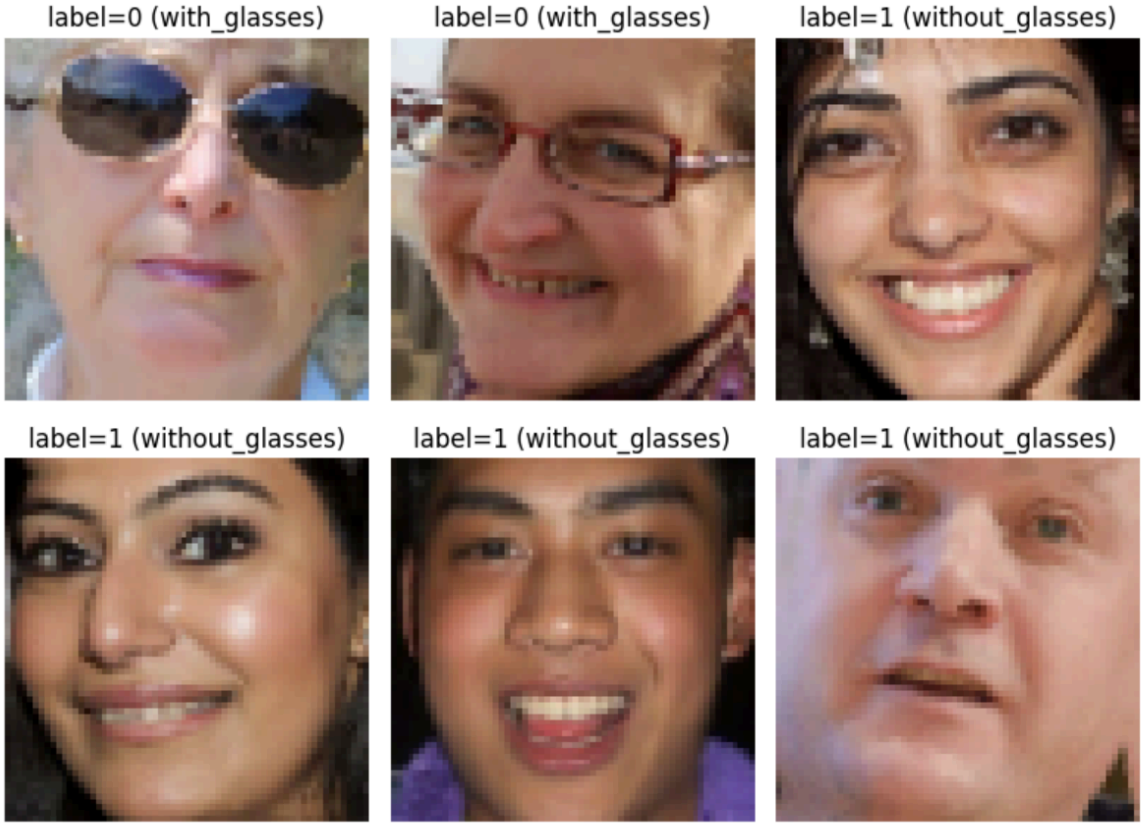
Una vez cargadas las imágenes, fue necesario aplicar un proceso de normalización. Las redes neuronales, especialmente aquellas que utilizan funciones de activación como ReLU, suelen entrenarse de manera más estable cuando los valores de entrada se encuentran en un rango acotado, normalmente entre 0 y 1. Para lograrlo se empleó una capa `Rescaling(1./255)`, la cual transforma cada píxel (originalmente entre 0 y 255) a un valor continuo entre 0 y 1. Este ajuste no altera la información visual relevante, pero sí facilita enormemente la convergencia durante el entrenamiento.

Sin embargo, la normalización por sí sola no es suficiente para garantizar un modelo robusto. En visión computacional es muy común que el modelo se sobreajuste si solo observa un conjunto limitado de variaciones entre las imágenes. Para mitigar este comportamiento se incorporó un módulo de *data augmentation*. Esta etapa añade transformaciones aleatorias que simulan variaciones que podrían ocurrir naturalmente en un entorno real, como pequeñas rotaciones o reflejos horizontales. En este proyecto se aplicaron dos modificaciones básicas pero muy efectivas: un giro horizontal aleatorio y una ligera rotación del 5%. Aunque estas

transformaciones puedan parecer simples, tienen un impacto considerable al permitir que el modelo observe un rango más amplio de configuraciones sin necesidad de añadir nuevas imágenes externas.

Es importante señalar que estas transformaciones se aplican únicamente al conjunto de entrenamiento. Los datasets de validación y prueba deben permanecer sin modificaciones, ya que representan el comportamiento real del modelo en imágenes que no han sido manipuladas artificialmente.

Aquí se pueden visualizar un ejemplo de las imágenes con la aplicación del data augmentation.



Finalmente, con el objetivo de optimizar aún más el rendimiento, los datasets fueron configurados utilizando cache() y prefetch(AUTOTUNE). Gracias a estas funciones, TensorFlow es capaz de almacenar una parte de los datos en memoria y preparar los siguientes lotes de imágenes incluso mientras el modelo se encuentra entrenando.

Y con todo lo anterior ya tendríamos nuestros datos preparados para pasar con el modelo, los cuales en esa ocasión son 2 modelos, los cuales pueden resolver la misma problemática sin embargo cada uno con un enfoque diferente.

MODELADO — CNN CONSTRUÍDA MANUALMENTE

Model: "sequential"

Layer (type)	Output Shape	Param #
data_augmentation (Sequential)	(None, 64, 64, 3)	0
conv2d (Conv2D)	(None, 64, 64, 16)	448
max_pooling2d (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_1 (Conv2D)	(None, 32, 32, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 64)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 32)	2,080
dense_1 (Dense)	(None, 1)	33

Total params: 25,697 (100.38 KB)
Trainable params: 25,697 (100.38 KB)
Non-trainable params: 0 (0.00 B)

Una arquitectura CNN es la que utilicé en esta ocasión para esta problemática. Este modelo busca capturar las características más relevantes del problema, la diferencia visual entre personas con lentes y personas sin lentes empleando una estructura sencilla pero funcional, adecuada para un problema de clasificación binaria.

La arquitectura implementada está compuesta por tres bloques convolucionales principales, seguidos de una etapa de agregación global y un pequeño clasificador denso.

1. Entrada y Data Augmentation Integrado

La red inicia con una capa input que define el tamaño de las imágenes procesadas (64×64×3). Inmediatamente después, se incorpora el módulo de *data augmentation*, el cual incluye transformaciones suaves como volteos horizontales y rotaciones leves. Estas variaciones permiten que el modelo observe un conjunto más amplio de configuraciones durante el entrenamiento.

Integrar el aumento de datos dentro del propio modelo, y no fuera del pipeline, tiene una ventaja importante: las transformaciones se aplican en GPU y de manera aleatoria en cada época, maximizando la eficiencia del proceso y aumentando la capacidad de generalización sin incrementar el tamaño del dataset.

2. Bloques Convolucionales

Cada uno de los tres bloques convolucionales sigue la estructura:

- Conv2D(filters, kernel_size=3, activation='relu', padding='same')
- MaxPooling2D()

Este patrón es uno de los más comunes en visión computacional, ya que combina extracción de características con reducción progresiva de resolución.

Conv2D con filtros 16 → 32 → 64

El incremento gradual en el número de filtros tiene una lógica fundamentada:

- 16 filtros en el primer bloque permiten detectar patrones básicos como bordes, líneas o contrastes simples.
  32 filtros en el segundo bloque capturan detalles más complejos, como siluetas, regiones brillantes u oclusiones parciales.
- 64 filtros en el tercer bloque aprenden características más abstractas, como formas globales del rostro, estructuras repetitivas o la presencia de lentes como objetos sólidos.

Esta progresión evita sobrecargar el modelo en capas iniciales y distribuye la capacidad de aprendizaje de forma más eficiente.

MaxPooling2D

Cada operación de *pooling* reduce las dimensiones espaciales de la imagen a la mitad, permitiendo que:

- el modelo se vuelva menos sensible a pequeñas traslaciones,
- disminuya el número de parámetros en las capas posteriores,
- se controle el riesgo de sobreajuste,
- y aumente la profundidad efectiva del modelo sin necesidad de matrices masivas.

Gracias a este patrón, la red logra entender tanto detalles finos como estructuras globales en los rostros.

3. Agregación y Clasificador Final

Luego de los bloques convolucionales, se aplica una capa GlobalAveragePooling2D(). Esta capa transforma los mapas de características en un vector compacto, tomando el promedio de cada filtro. A diferencia de un Flatten tradicional, la agregación global:

- reduce drásticamente el número de parámetros,
- hace al modelo menos propenso a sobreajustar,
- y conecta de manera más directa cada filtro aprendido con la clasificación final.

Posteriormente se incluye una capa Dropout(0.2), la cual desactiva aleatoriamente el 20% de las neuronas durante el entrenamiento. Esto actúa como regularizador y evita que el modelo memorice patrones específicos de entrenamiento.

Finalmente, el clasificador consiste en:

- un Dense(32, activation='relu') para combinar y procesar las características extraídas,
- y un último Dense(1, activation='sigmoid'), encargado de producir una probabilidad entre 0 y 1 correspondiente a la clase *con lentes* o *sin lentes*.

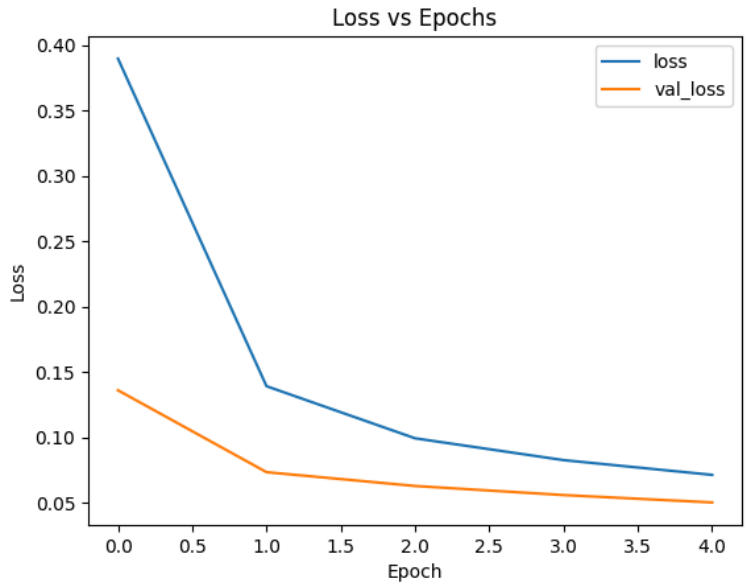
Dado que el problema es estrictamente binario, la activación sigmoide y la función de pérdida binary_crossentropy constituyen la elección correcta y estándar en la literatura.

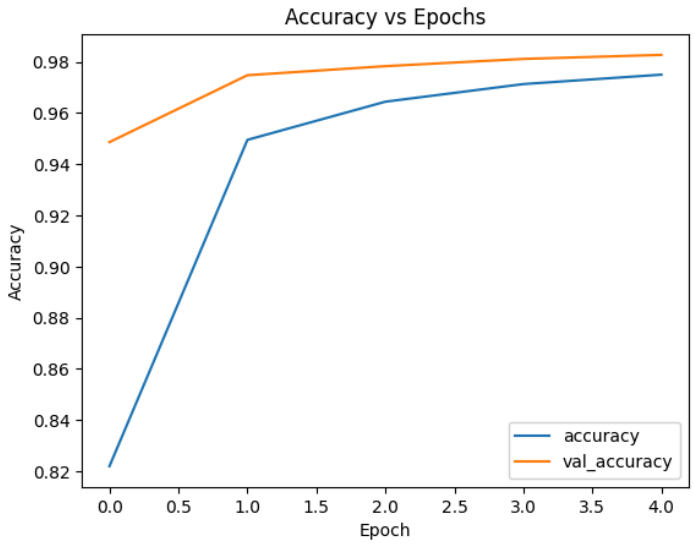
4. Entrenamiento

El modelo fue entrenado durante 5 épocas utilizando:

- Adam como optimizador, seleccionado por su estabilidad y capacidad de adaptarse dinámicamente al gradiente.
- EarlyStopping, que detiene el entrenamiento si la pérdida de validación deja de mejorar, evitando el sobreajuste.
  ReduceLROnPlateau, que reduce automáticamente la tasa de aprendizaje cuando el progreso se estanca.
- ModelCheckpoint, el cual almacena únicamente los pesos del mejor modelo obtenido en entrenamiento.

Posteriormente, se trazaron curvas de *loss* y *accuracy* que permiten inspeccionar visualmente el desempeño del modelo a lo largo de las épocas. Finalmente, se cargó el mejor checkpoint para evaluar su rendimiento real.





Como se puede observar en pocas épocas el modelo alcanza un accuracy significativo por lo que podemos decir que esta arquitectura funcionó con excelentes resultados. Tanto en validation como training se observan valores muy similares por lo que no existe overffiting.

MODELADO — MobileNetV2 (Transfer Learning)

9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

Model: "sequential_1"

Layer (type)	Output Shape	Param #
data_augmentation (Sequential)	(None, 64, 64, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 2, 2, 1280)	2,257,984
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1280)	0
dropout_1 (Dropout)	(None, 1280)	0
dense_2 (Dense)	(None, 128)	163,968
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129

Tras entrenar el modelo CNN construido manualmente, el siguiente paso consistió en implementar una arquitectura más robusta y moderna: MobileNetV2. Esta red pertenece a la familia de modelos diseñados para visión computacional eficiente, y destaca por ofrecer un rendimiento muy competitivo con un número reducido de parámetros, lo cual la hace ideal para tareas de clasificación de imágenes en entornos con recursos limitados, como dispositivos móviles o aplicaciones en tiempo real.

En lugar de entrenar un modelo desde cero, se optó por un enfoque de Transfer Learning, aprovechando pesos previamente entrenados sobre el dataset ImageNet, el cual contiene más de un millón de imágenes y mil categorías diferentes. Este preentrenamiento permite que MobileNetV2 ya posea conocimiento previo sobre texturas, bordes, formas y patrones visuales generales, lo que facilita que el modelo se adapte rápidamente a una nueva tarea como la clasificación de rostros con o sin lentes.

Para garantizar consistencia con el modelo previo, se reutilizó exactamente el mismo pipeline de datos: mismas imágenes, mismos tamaños (64×64), normalización, data augmentation y batching. La diferencia esencial radica en la arquitectura del modelo

```
base_model = MobileNetV2(
    input_shape=INPUT_SHAPE,
    include_top=False,
    weights="imagenet"
)

base_model.trainable = False  # Congelamos la base convolucional
```

A partir de esta base, el modelo completo se construyó añadiendo un pequeño clasificador:

```
model = Sequential([
    Input(shape=INPUT_SHAPE),
    data_augmentation,          # Tus aumentos leves
    base_model,                 # MobileNetV2 congelado
    GlobalAveragePooling2D(),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')  # Clasificación binaria
])
```

Transfer Learning:
Aprovecha características previamente aprendidas, evitando tener que entrenar millones de parámetros desde cero.

Congelamiento inicial:
Esto evita sobreajuste temprano y acelera el entrenamiento. El modelo aprende rápidamente gracias a que solo se entrena la parte final (clasificador).

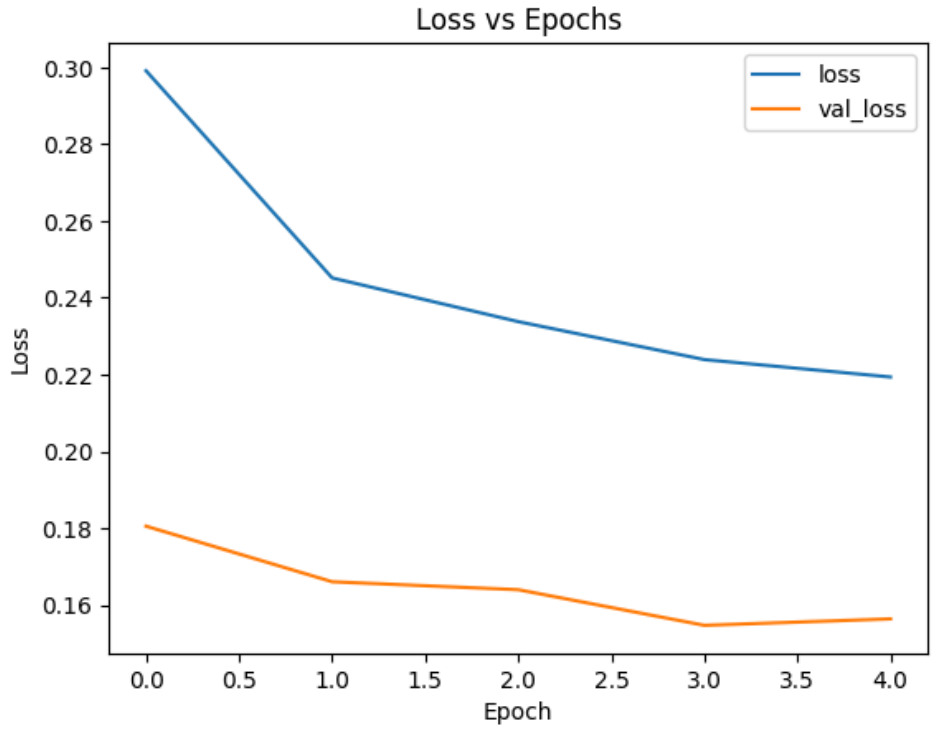
GlobalAveragePooling2D:
Reemplaza las capas densas tradicionales basadas en flattening, reduciendo parámetros y evitando sobreajuste.

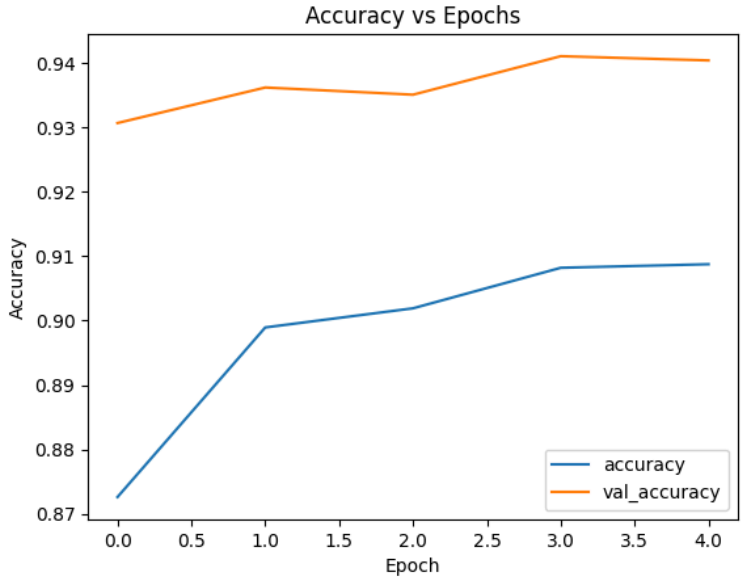
Dropout del 30% y 20%:
Reduce el riesgo de memorizar ejemplos específicos del dataset.

Capa densa de 128 neuronas:
Actúa como adaptador entre las características generadas por MobileNetV2 y la tarea específica de clasificación binaria.

Salida con Sigmoid:
Devuelve una probabilidad entre 0 y 1, ideal para distinguir “con lentes” vs “sin lentes”.

Resultados:





A comparación con mi CNN tal parece ser que este modelo pre entrenado no generaliza bien, teniendo menos eficiencia y teniendo una brecha considerable con respecto a su entrenamiento.

COMPARACIÓN ENTRE CNN MANUAL Y MobileNetV2

1. Diferencias arquitectónicas fundamentales

CNN Manual

- Construida completamente desde cero.
  Consta de 3 bloques convolucionales con 16, 32 y 64 filtros.
- Extracción de características limitada a patrones simples (bordes, texturas básicas).
- Número total de parámetros reducido.
- Riesgo mayor de underfitting en datasets complejos.
- Su rendimiento depende exclusivamente de lo aprendido en este entrenamiento específico.

MobileNetV2

- Arquitectura moderna diseñada para dispositivos móviles.
- Utiliza Transfer Learning con pesos pre-entrenados en ImageNet.
- Extrae características complejas desde el inicio (formas, contornos, patrones visuales de alto nivel).
- Congelación del backbone → el modelo evita sobreajuste temprano.
- El clasificador final (densas) es pequeño y eficiente.
- Generaliza mejor incluso en casos donde la variabilidad del dataset es alta.

Conclusión:
La CNN manual es simple; MobileNetV2 es una arquitectura profesional optimizada para visión computacional avanzada.

CONCLUSIONES FINALES

El desarrollo de este proyecto permitió comprender de manera práctica cómo arquitecturas de deep learning pueden ser aplicadas a problemas reales de visión computacional, en este caso, la clasificación de rostros para determinar si una persona porta lentes o no. A lo largo del proceso se construyeron dos modelos con enfoques distintos: una CNN diseñada manualmente desde cero y un modelo basado en MobileNetV2 empleando Transfer Learning. Ambos fueron entrenados bajo las mismas condiciones, lo que permitió realizar una comparación justa y fundamentada.

En primer lugar, la CNN manual demostró ser una arquitectura funcional, capaz de aprender patrones relevantes y obtener un rendimiento sólido. Su implementación resultó útil para comprender cómo operan los bloques convolucionales, así como la importancia de elementos como la activación ReLU, el pooling y los regularizadores.