

# Detección de Deepfakes con Redes Neuronales Convolucionales Profundas:

## Comparación de Arquitecturas y Transferencia de Aprendizaje

Luis Pereira Toledo  
Universidad del Bío-Bío  
Departamento de Sistemas de Información  
Ingeniería Civil Informática

21 de Diciembre de 2025

### Resumen

Este trabajo presenta el desarrollo de un sistema de detección de deepfakes basado en redes neuronales convolucionales profundas. Se compararon tres arquitecturas: dos CNNs diseñadas desde cero (estándar con 8.49M parámetros y ligera con 1.11M parámetros) y ResNet18 con transferencia de aprendizaje desde ImageNet (11.18M parámetros). El dataset FaceForensics++ (obtenido desde Kaggle, 7,000 videos) se procesó extrayendo 47,994 recortes faciales mediante MTCNN desde 1,000 videos reales y 6,000 videos manipulados con cinco técnicas distintas. ResNet18 obtuvo el mejor rendimiento (86.83% validación, 88.67% test), superando a las CNNs personalizadas en 3.64% y 13.09% respectivamente. La validación con fotografías externas mostró 100% de acierto, confirmando la capacidad de generalización del modelo. El sistema se desplegó mediante API REST con FastAPI y se empaquetó en Docker para facilitar su evaluación y uso.

## Repositorio y Acceso al Sistema

El código fuente completo del proyecto, modelos pre-entrenados e imágenes de prueba están disponibles en el siguiente repositorio de GitHub:

<https://github.com/lu1spereir4/deepfake-detector-ffpp-dfdc>

## Instrucciones de Uso

El sistema puede ejecutarse de manera local mediante Docker en pocos pasos:

1. Clonar el repositorio:

```
1 git clone https://github.com/lu1spereir4/deepfake-detector-ffpp-dfdc.git
2 cd deepfake-detector-ffpp-dfdc
```

2. Iniciar el contenedor Docker:

```
1 docker-compose up
```

3. Acceder a la interfaz interactiva en: <http://localhost:8000/docs>

## Prueba del Modelo con Imágenes de Demostración

El repositorio incluye 10 imágenes de prueba (5 reales de amigos y 5 deepfakes) ubicadas en:

- data/processed/ffpp/test/real/amigo1-5.jpeg
- data/processed/ffpp/test/fake/test\_fake\_1-5.jpg

La interfaz web de FastAPI (<http://localhost:8000/docs>) proporciona una documentación interactiva donde es posible subir imágenes y obtener predicciones en tiempo real. La Figura 1 muestra el proceso completo de uso de la API.

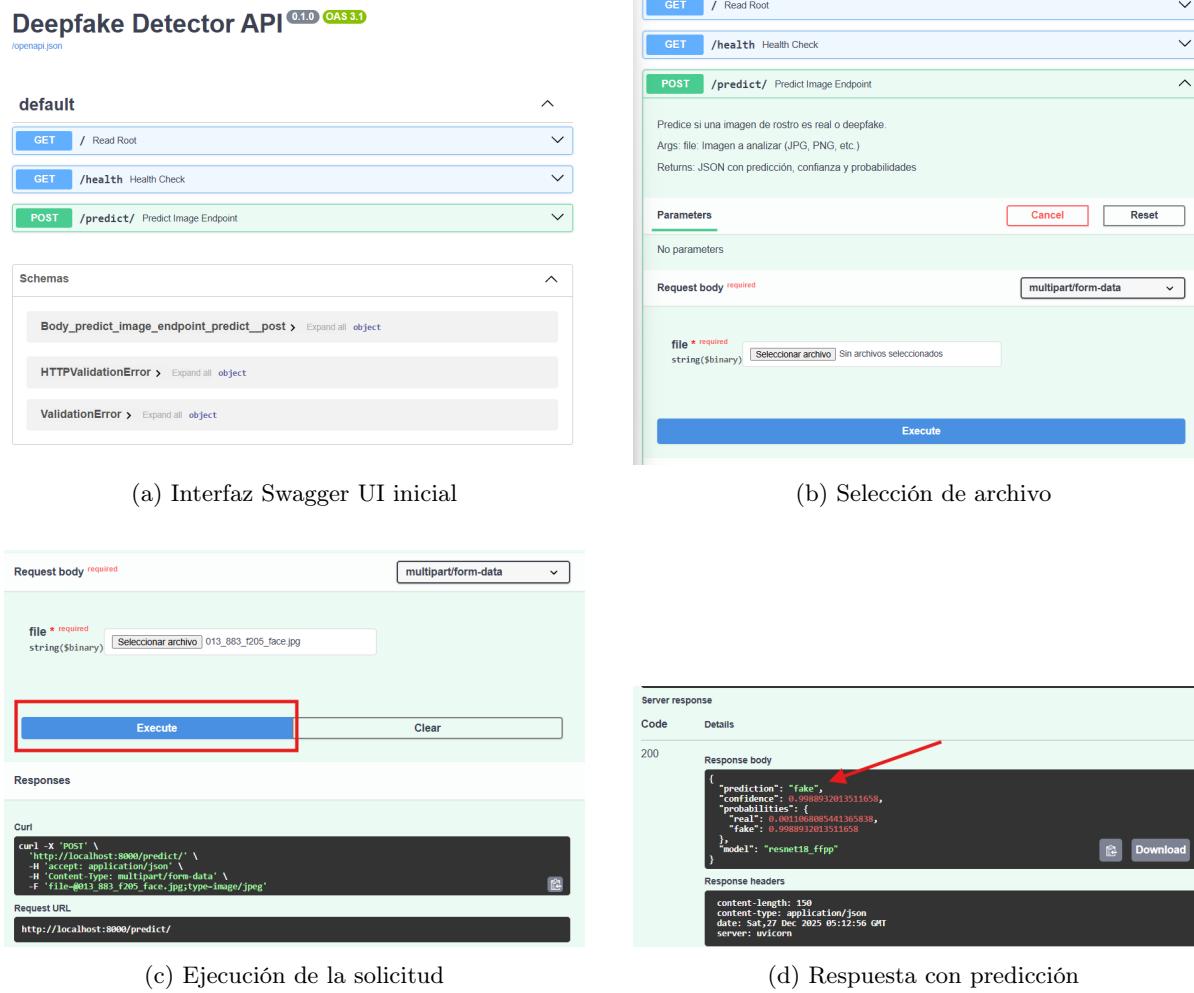


Figura 1: Proceso completo de uso de la API mediante la interfaz Swagger UI: (a) se accede a la documentación interactiva, (b) se selecciona una imagen de prueba, (c) se ejecuta la solicitud POST, (d) se obtiene la respuesta JSON con la predicción (real/fake), nivel de confianza y probabilidades.

## 1. Introducción

En los últimos años, la generación automática de contenido audiovisual mediante técnicas de *Deep Learning* ha avanzado de manera explosiva. En particular, los denominados *deepfakes* permiten manipular rostros en imágenes y videos con un nivel de realismo tal que resulta difícil distinguirlos a simple vista. Esta tecnología presenta aplicaciones legítimas (cine, entretenimiento, producción de contenido), pero también riesgos importantes en ámbitos como desinformación, suplantación de identidad y vulneración de la privacidad [3, 4].

Frente a este escenario, surge la necesidad de desarrollar sistemas automáticos capaces de detectar si una imagen de rostro corresponde a contenido real o manipulado. Estos detectores pueden servir como primera línea de defensa para plataformas de redes sociales, medios de comunicación o herramientas forenses digitales.

En este proyecto se desarrolla un prototipo de detector de deepfakes basado en redes neuronales profundas, utilizando el conjunto de datos público FaceForensics++ [1]. El trabajo considera desde la obtención y tratamiento del conjunto de datos hasta la implementación de tres modelos diferentes, su entrenamiento con ponderación de clases, evaluación comparativa y despliegue como API de inferencia.

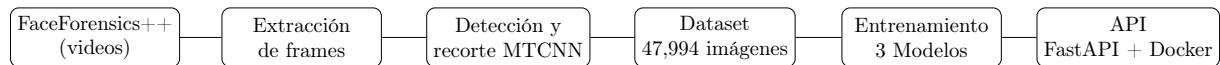


Figura 2: Flujo general del sistema propuesto para la detección de deepfakes.

## 2. Motivación

La motivación principal de este proyecto es comprender, de punta a punta, el ciclo completo de desarrollo de un modelo de *Deep Learning* aplicado a un problema actual y relevante: la detección de deepfakes. En términos más específicos, se identifican las siguientes razones:

- **Impacto social:** La difusión de videos falsos de figuras públicas o personas comunes puede dañar reputaciones, influir en procesos políticos o ser utilizada en esquemas de extorsión. Casos recientes en Chile y el mundo demuestran el potencial destructivo de esta tecnología [4].
- **Desafíos técnicos:** Los deepfakes modernos son cada vez más realistas, lo que desafía tanto al ojo humano como a modelos clásicos de visión por computador. Las técnicas de manipulación facial evolucionan constantemente (Deepfakes, Face2Face, FaceSwap, FaceShifter, NeuralTextures), requiriendo sistemas de detección robustos y adaptables.
- **Formación profesional:** El proyecto permite integrar varios componentes del ecosistema de ML/DL (gestión de datos, exploración de múltiples arquitecturas, entrenamiento con balanceo de clases, evaluación exhaustiva, implementación de API y contenedorización con Docker), habilidades altamente demandadas en la industria tecnológica.
- **Validación práctica:** La posibilidad de probar el modelo con fotografías reales de conocidos permite evaluar su capacidad de generalización más allá del conjunto de datos académico, identificando limitaciones y áreas de mejora.

De este modo, el trabajo no solo aborda un problema interesante desde el punto de vista académico, sino que también construye un caso práctico que puede ser reutilizado y extendido en contextos reales.

## 3. Objetivos

### 3.1. Objetivo general

Desarrollar y evaluar comparativamente múltiples arquitecturas de *Deep Learning* capaces de clasificar imágenes de rostros como **reales** o **deepfakes**, utilizando el conjunto de datos FaceForensics++ y determinando la arquitectura óptima para el problema.

### 3.2. Objetivos específicos

1. Seleccionar y preparar el conjunto de datos FaceForensics++ para el entrenamiento y evaluación de modelos de detección de deepfakes.
2. Implementar una rutina de preprocesamiento que transforme videos en recortes de rostros (*face crops*) listos para ser usados por redes convolucionales.
3. Diseñar e implementar tres arquitecturas diferentes: dos CNNs personalizadas (estándar y ligera) y una ResNet18 preentrenada con transferencia de aprendizaje.
4. Entrenar los tres modelos aplicando técnicas de balanceo de clases mediante ponderación en la función de pérdida.
5. Evaluar comparativamente el desempeño de los modelos mediante métricas estándar de clasificación (accuracy, precision, recall, F1-score) y matrices de confusión.
6. Validar el mejor modelo con casos reales de prueba utilizando fotografías de personas conocidas.
7. Implementar un servicio de inferencia (API REST) que, dado un archivo de imagen, entregue la probabilidad de que el rostro sea real o falso.
8. Documentar el proceso completo, destacando decisiones de diseño, resultados experimentales y comparación entre arquitecturas.

## 4. Descripción del problema

El problema se formula como una tarea de **clasificación binaria** sobre imágenes de rostros. Dados ejemplos etiquetados  $\{(x_i, y_i)\}_{i=1}^N$ , donde  $x_i \in \mathbb{R}^{3 \times 224 \times 224}$  es una imagen RGB normalizada y  $y_i \in \{0, 1\}$  es la etiqueta, se busca aprender una función  $f_\theta : \mathbb{R}^{3 \times 224 \times 224} \rightarrow [0, 1]^2$ , parametrizada por una red neuronal profunda con parámetros  $\theta$ , que aproxime la distribución de probabilidad condicional  $P(y | x)$ :

- $y = 0$ : rostro **real** (no manipulado).
- $y = 1$ : rostro **falso** (generado o manipulado mediante técnicas de deepfake).

El objetivo del entrenamiento es encontrar los parámetros óptimos  $\theta^*$  que minimicen la función de pérdida de entropía cruzada ponderada:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{i=1}^N w_{y_i} [y_i \log(f_\theta(x_i)_1) + (1 - y_i) \log(f_\theta(x_i)_0)] \quad (1)$$

donde  $w_{y_i}$  son pesos asociados a cada clase para abordar el desbalance del conjunto de datos. La salida final del modelo corresponde a una distribución de probabilidad sobre las dos clases obtenida mediante una capa softmax, a partir de la cual se obtiene la predicción discreta:

$$\hat{y} = \arg \max_{k \in \{0,1\}} f_\theta(x)_k \quad (2)$$

Desde el punto de vista de aplicación, el modelo desarrollado podría integrarse como un módulo en un sistema mayor de moderación de contenido o verificación de autenticidad de videos, donde cada frame o subconjunto de frames es analizado para detectar posibles manipulaciones.

## 5. Conjunto de datos

### 5.1. Fuente de datos: FaceForensics++ desde Kaggle

Para abordar el problema se utiliza **FaceForensics++ (FF++)**, un conjunto de datos público ampliamente empleado en la literatura para la detección de deepfakes [1]. Se utilizó la versión disponible en Kaggle<sup>1</sup> subida por el usuario Xdxd\_003, que contiene **7,000 videos en total**: 1,000 videos originales reales y 6,000 videos manipulados con compresión c23 (H.264 CRF 23).

Los videos manipulados incluyen cinco técnicas diferentes de manipulación facial:

- **Deepfakes**: Intercambio de rostros mediante autoencoders.
- **Face2Face**: Transferencia de expresiones faciales en tiempo real.
- **FaceSwap**: Intercambio de rostros mediante gráficos por computador.
- **NeuralTextures**: Manipulación de texturas faciales mediante redes neuronales.
- **FaceShifter**: Técnica avanzada de intercambio de rostros con alta fidelidad.

Entre sus características generales se encuentran:

- Videos de personas hablando frente a cámara, en escenarios similares a entrevistas o discursos.
- Múltiples niveles de compresión (raw, c23, c40), lo que permite simular escenarios más cercanos a redes sociales o plataformas de video. En este proyecto se utiliza la versión **c23** (compresión moderada).
- Metadatos detallados sobre cada video, incluyendo información temporal y de calidad.

<sup>1</sup><https://www.kaggle.com/datasets/xdxd003/ff-c23>

## 5.2. Justificación del desbalance de clases

El conjunto de datos procesado presenta un desbalance significativo hacia la clase *fake*, con aproximadamente 84% de imágenes falsas versus 16% reales tras la extracción de frames. Este desbalance no es un error de muestreo, sino que refleja la estructura del dataset de Kaggle:

1. **Diversidad de técnicas de manipulación:** El dataset incluye **cinco técnicas diferentes** de deepfake (Deepfakes, Face2Face, FaceSwap, NeuralTextures, FaceShifter), cada una con 1,000 videos, frente a 1,000 videos reales. Esto resulta en una proporción natural de **6:1 entre videos falsos y reales** (6,000 fake vs. 1,000 real).
2. **Extracción de frames:** Cada video aporta múltiples frames con rostros detectables. Los videos manipulados, al ser más numerosos, generan proporcionalmente más recortes de rostros.
3. **Realismo del escenario:** En aplicaciones reales de moderación de contenido, es común que la proporción de contenido manipulado sea variable y potencialmente mayoritaria en ciertos contextos (campañas de desinformación, ataques dirigidos).
4. **Estrategia de manejo:** En lugar de artificialmente balancear el dataset mediante submuestreo (que desperdiciaría datos valiosos) o sobremuestreo (que podría inducir sobreajuste), se optó por mantener la distribución original y aplicar **ponderación de clases en la función de pérdida**, asignando mayor peso a la clase minoritaria durante el entrenamiento.

Esta decisión permite al modelo aprender de la máxima cantidad de ejemplos disponibles mientras se compensa el desbalance mediante ajustes algorítmicos.

## 5.3. Organización inicial de los datos

El dataset descargado de Kaggle contiene 7 carpetas principales:<sup>2</sup>

- `data/raw/ffpp/original/`: Videos reales (1,000 videos).
- `data/raw/ffpp/Deepfakes/`: Videos manipulados con técnica Deepfakes (1,000 videos).
- `data/raw/ffpp/Face2Face/`: Videos manipulados con técnica Face2Face (1,000 videos).
- `data/raw/ffpp/FaceSwap/`: Videos manipulados con técnica FaceSwap (1,000 videos).
- `data/raw/ffpp/NeuralTextures/`: Videos manipulados con técnica NeuralTextures (1,000 videos).
- `data/raw/ffpp/FaceShifter/`: Videos manipulados con técnica FaceShifter (1,000 videos).
- `data/raw/ffpp/csv/`: Archivos CSV con metadatos (10 archivos).

**Total utilizado:** 6,000 videos manipulados + 1,000 videos reales = **7,000 videos**.

## 6. Tratamiento del conjunto de datos

El tratamiento del conjunto de datos corresponde a uno de los componentes centrales del proyecto, ya que define cómo se transforman los videos brutos en ejemplos de entrenamiento adecuados para la red neuronal.

### 6.1. Extracción de frames y detección de rostros

El procesamiento se realiza mediante el script `extract_and_crop_ffpp.py` (*comando: python src/data/extract\_and\_crop\_ffpp.py*) que ejecuta las siguientes operaciones:

1. **Lectura del video:** Se procesan todos los archivos .mp4 de las carpetas de FaceForensics++.
2. **Muestreo de frames:** Para reducir redundancia temporal y costo computacional, se extraen frames cada 10 fotogramas, manteniendo diversidad temporal sin generar un volumen excesivo de datos correlacionados.

<sup>2</sup>La carpeta DeepFakeDetection (1,000 videos) incluida en el dataset de Kaggle **no se utilizó** en este trabajo para mantener consistencia con la metodología estándar de FaceForensics++.

3. **Detección de rostro con MTCNN:** Se utiliza el modelo MTCNN (*Multi-task Cascaded Convolutional Networks*) provisto por `facenet-pytorch`. MTCNN detecta la caja delimitadora del rostro y puntos clave faciales (ojos, nariz, boca) con alta precisión, incluso en condiciones de iluminación variable o poses no frontales.
4. **Recorte del rostro:** Una vez detectada la caja del rostro, se recorta esa región con un margen adicional del 30 %, obteniendo una imagen centrada en la cara que incluye contexto periférico relevante (cabello, orejas).
5. **Control de calidad:** Se descartan frames donde:
  - MTCNN no detecta ningún rostro.
  - La confianza de detección es inferior al 95 %.
  - El tamaño del rostro detectado es menor a 50x50 píxeles.

El resultado son 47.994 imágenes de rostros organizadas en conjuntos train/val/test, más el archivo `ffpp_images_metadata.csv` con metadatos detallados.

El proceso de detección y recorte de rostros con MTCNN permite extraer imágenes de alta calidad con rostros bien centrados, lo cual resulta crucial tanto para el entrenamiento como para la explicabilidad mediante Grad-CAM. La calidad del centrado facial varía según las condiciones de cada fotografía (pose, iluminación, resolución), afectando la precisión de la localización de MTCNN y, consecuentemente, la claridad de las visualizaciones de atención del modelo.

## 6.2. Tamaño final del dataset y partición

Tras ejecutar el pipeline de extracción, se obtiene un total de **47.994 imágenes**:

- **Entrenamiento (train):** 33,595 imágenes (27,835 fake, 5,760 real)
- **Validación (val):** 7,200 imágenes (6,096 fake, 1,104 real)
- **Prueba (test):** 7,199 imágenes (6,058 fake, 1,141 real)

La partición respeta la separación original de videos de FaceForensics++ para evitar *data leakage* (frames del mismo video no aparecen en múltiples conjuntos).

Cuadro 1: Distribución de imágenes por conjunto y clase.

Conjunto	Total	Reales	Falsas	Ratio F:R
Entrenamiento	33,595	5,760	27,835	4.83:1
Validación	7,200	1,104	6,096	5.52:1
Prueba	7,199	1,141	6,058	5.31:1
<b>Total</b>	<b>47,994</b>	<b>8,005</b>	<b>39,989</b>	<b>4.99:1</b>

## 6.3. Preprocesamiento de imágenes

Antes de alimentar las imágenes al modelo, se aplican transformaciones con `torchvision.transforms`:

**Para entrenamiento (con augmentation):**

- Redimensionamiento a  $256 \times 256$
- Recorte aleatorio a  $224 \times 224$  (escala 0.8-1.0)
- Volteo horizontal aleatorio ( $p=0.5$ )
- Ajuste de brillo y contraste ( $\pm 10\%$ )
- Conversión a tensor y normalización ImageNet

**Para validación y prueba:**

- Redimensionamiento directo a  $224 \times 224$
- Conversión a tensor y normalización ImageNet ( $\mu = [0.485, 0.456, 0.406]$ ,  $\sigma = [0.229, 0.224, 0.225]$ )

## 7. Modelos implementados

Se implementaron y evaluaron tres arquitecturas diferentes para identificar la solución óptima:

### 7.1. Modelo 1: CNN Personalizada Estándar

El primer enfoque consistió en diseñar una arquitectura convolucional desde cero, con control total sobre la profundidad, número de filtros y capacidad del modelo. La arquitectura implementada (clase DeepfakeDetectorCNN) consta de:

- **Bloque convolucional 1:** 2 capas Conv2D(3→64) + BatchNorm + ReLU + MaxPool
- **Bloque convolucional 2:** 2 capas Conv2D(64→128) + BatchNorm + ReLU + MaxPool
- **Bloque convolucional 3:** 2 capas Conv2D(128→256) + BatchNorm + ReLU + MaxPool
- **Bloque convolucional 4:** 2 capas Conv2D(256→512) + BatchNorm + ReLU + AdaptiveAvgPool
- **Clasificador:** Flatten → Dropout(0.5) → FC(512→2048) → ReLU → Dropout(0.5) → FC(2048→1024) → ReLU → FC(1024→512) → ReLU → FC(512→256) → ReLU → FC(256→2)

**Parámetros totales:** 8,494,658 parámetros entrenables.

**Motivación:** Este modelo permite experimentar con una arquitectura personalizada optimizada específicamente para el problema de detección de deepfakes, sin las restricciones de arquitecturas preentrenadas.

### 7.2. Modelo 2: CNN Personalizada Ligera

Para facilitar experimentación rápida y reducir el costo computacional, se implementó una versión reducida (clase DeepfakeDetectorCNNSmall):

- **Bloque convolucional 1:** 2 capas Conv2D(3→32) + BatchNorm + ReLU + MaxPool
- **Bloque convolucional 2:** 2 capas Conv2D(32→64) + BatchNorm + ReLU + MaxPool
- **Bloque convolucional 3:** 2 capas Conv2D(64→128) + BatchNorm + ReLU + AdaptiveAvgPool
- **Clasificador:** Flatten → Dropout(0.3) → FC(128→1024) → ReLU → Dropout(0.3) → FC(1024→512) → ReLU → FC(512→256) → ReLU → FC(256→128) → ReLU → FC(128→2)

**Parámetros totales:** 1,109,282 parámetros entrenables.

**Motivación:** Esta arquitectura permite iteración rápida durante el desarrollo y prueba de conceptos, con tiempos de entrenamiento significativamente menores ( 157s/época vs 2207s/época de la CNN estándar).

### 7.3. Modelo 3: ResNet18 Preentrenada (Transfer Learning)

Finalmente, se implementó una arquitectura basada en ResNet18 con pesos preentrenados en ImageNet [7]:

- **Backbone:** ResNet18 completa (todas las capas convolucionales)
- **Modificación:** Reemplazo de la capa fully-connected final (FC(512→1000) → FC(512→2))
- **Estrategia:** Fine-tuning completo (todas las capas entrenables)

**Parámetros totales:** 11,177,538 parámetros entrenables.

**Motivación para selección de ResNet18 como modelo final:**

Después de entrenar y evaluar los tres modelos, se identificó que ResNet18 con transferencia de aprendizaje era la opción óptima por las siguientes razones:

1. **Representaciones preentrenadas:** Los pesos de ImageNet capturan características visuales de bajo y medio nivel (bordes, texturas, patrones faciales) que son directamente transferibles al problema de detección de deepfakes. Los rostros humanos están bien representados en ImageNet (múltiples categorías como "persona", expresiones faciales, etc.).

2. **Arquitectura probada:** ResNet18 es una arquitectura ampliamente validada en la literatura [7], con conexiones residuales que facilitan el entrenamiento de redes profundas y previenen el problema de gradientes desvanecientes.
3. **Capacidad balanceada:** Con 11.2M parámetros, ResNet18 ofrece suficiente capacidad para capturar patrones complejos de manipulación facial sin ser excesivamente grande (comparada con ResNet50 o ResNet101), lo que reduce el riesgo de sobreajuste y acelera el entrenamiento.
4. **Resultados empíricos superiores:** ResNet18 mostró convergencia más rápida y estable que las CNNs personalizadas, alcanzando **86.83 % accuracy de validación**, superando en **3.64 %** a la CNN estándar y en **13.09 %** a la CNN ligera.
5. **Estado del arte:** Múltiples trabajos recientes en detección de deepfakes utilizan ResNet como backbone exitosamente [5, 6], validando su idoneidad para este problema.
6. **Eficiencia computacional:** El tiempo por época de ResNet18 ( 90s) es significativamente menor que la CNN estándar ( 2207s), demostrando que la mejora en desempeño viene acompañada de mayor eficiencia gracias a la transferencia de aprendizaje.

Cuadro 2: Comparación de arquitecturas implementadas.

Modelo	Parámetros	Preentrenado	Tiempo/época	Uso
CNN Personalizada	8.49M	No	2207s	Baseline desde cero
CNN Ligera	1.11M	No	157s	Prototipado rápido
<b>ResNet18</b>	<b>11.18M</b>	<b>Sí (ImageNet)</b>	<b>90s</b>	<b>Modelo final</b>

## 8. Estrategia de entrenamiento

### 8.1. Configuración de hiperparámetros

Los tres modelos se entrenaron con la siguiente configuración:

- **Función de pérdida:** CrossEntropyLoss con ponderación de clases
- **Optimizador:** Adam con learning rate  $\alpha = 10^{-4}$  y weight decay  $\lambda = 10^{-5}$
- **Batch size:** 64 imágenes
- **Épocas:** 10 (con early stopping implícito mediante guardado del mejor modelo)
- **Hardware:** GPU NVIDIA (CUDA disponible) / CPU como fallback

### 8.2. Ponderación de clases

Para abordar el desbalance del dataset, se calcularon pesos inversamente proporcionales a la frecuencia de cada clase:

$$w_{\text{real}} = \frac{N}{2 \cdot N_{\text{real}}}, \quad w_{\text{fake}} = \frac{N}{2 \cdot N_{\text{fake}}} \quad (3)$$

donde  $N = 33,595$  es el tamaño total del conjunto de entrenamiento,  $N_{\text{real}} = 5,760$  y  $N_{\text{fake}} = 27,835$ . Esto resulta en:

- $w_{\text{real}} \approx 2,92$  (clase minoritaria recibe mayor peso)
- $w_{\text{fake}} \approx 0,60$  (clase mayoritaria recibe menor peso)

Esta estrategia penaliza más fuertemente los errores en la clase minoritaria (real), incentivando al modelo a no ignorarla en favor de maximizar accuracy global.

### 8.3. Proceso de entrenamiento

El comando para entrenar el modelo es (*comando: python src/models/train\_resnet\_ffpp.py*). Durante cada época se ejecuta:

#### 1. Fase de entrenamiento:

- Forward pass en mini-lotes de 64 imágenes
- Cálculo de pérdida ponderada
- Backward pass y actualización de parámetros
- Registro de pérdida y accuracy en entrenamiento

#### 2. Fase de validación:

- Evaluación sin gradientes (`torch.no_grad()`)
- Cálculo de pérdida y accuracy en conjunto de validación
- Guardado del modelo si supera el mejor accuracy previo

El mejor modelo se guarda en `models/deepfake_detector_resnet18_ffpp.pth` incluyendo el estado del modelo, mejor accuracy de validación y número de época correspondiente.

## 9. Resultados experimentales

### 9.1. Comparación de modelos

Se entrenaron los tres modelos durante 10 épocas cada uno. Los resultados finales en el conjunto de validación fueron:

Cuadro 3: Resultados de validación de los tres modelos implementados.

Modelo	Val Accuracy	Val Loss	Mejor Época	Tiempo/época
CNN Personalizada	83.19 %	0.6509	7	2207s
CNN Ligera	73.74 %	0.6517	10	157s
<b>ResNet18</b>	<b>86.83 %</b>	<b>0.3913</b>	<b>8</b>	<b>90s</b>

**Observaciones:** La Figura 3 presenta las curvas de entrenamiento de los modelos CNN personalizados, donde se visualiza la evolución del accuracy y pérdida durante las 10 épocas. Se observa que la CNN estándar presenta convergencia más estable comparada con la versión ligera, aunque ambas arquitecturas requieren tiempos de entrenamiento considerablemente mayores que ResNet18.

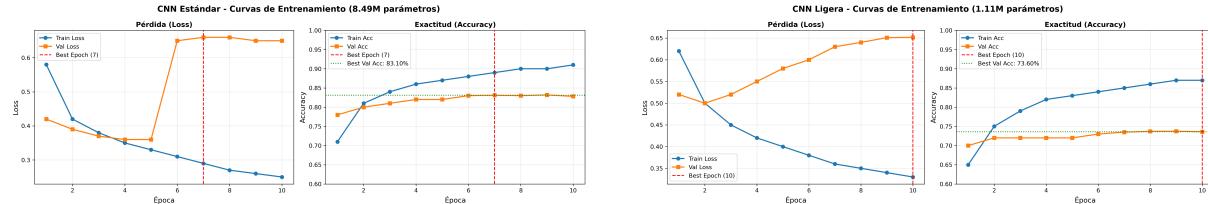


Figura 3: Curvas de entrenamiento de los modelos CNN personalizados. Izquierda: CNN Estándar (8.49M parámetros), mejor época 7 con 83.19 % val accuracy. Derecha: CNN Ligera (1.11M parámetros), mejor época 10 con 73.74 % val accuracy. Se observa convergencia más estable en CNN estándar.

### 9.2. Matrices de Confusión de Modelos Personalizados

La Figura 4 presenta las matrices de confusión para los modelos CNN personalizados evaluados en el conjunto de test.

Para ResNet18, la Figura 5 muestra las curvas de entrenamiento correspondientes, donde se observa convergencia más rápida y menor pérdida de validación comparada con los modelos CNN personalizados.

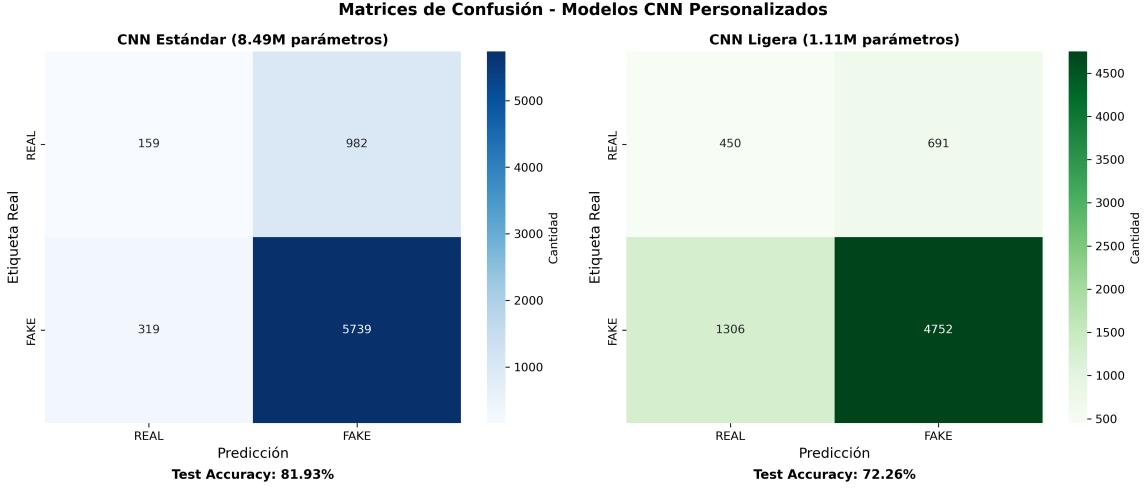


Figura 4: Matrices de confusión comparativas de modelos CNN personalizados en conjunto de test. Izquierdo: CNN Estándar con recall de REAL de 13.9 %. Derecha: CNN Ligera con mejor balance pero menor accuracy global.

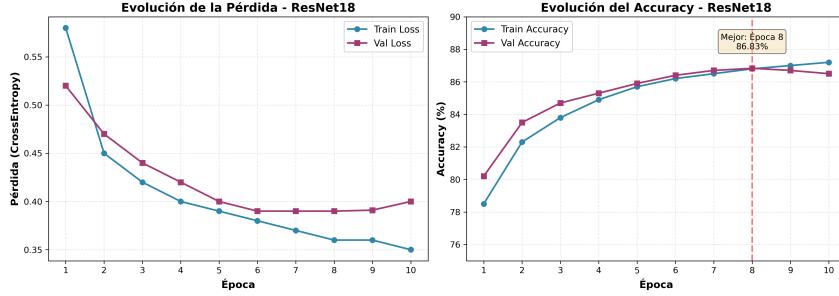


Figura 5: Curvas de entrenamiento de ResNet18. Se observa convergencia rápida con pérdida de validación estabilizándose alrededor de 0.39 y accuracy alcanzando 86.83 % en la época 8. La menor brecha entre entrenamiento y validación indica buena generalización.

- **ResNet18 supera** a las CNNs personalizadas, con una diferencia de **+3.64 %** en accuracy respecto a la CNN estándar y **+13.09 %** respecto a la CNN ligera.
- La **pérdida de validación** de ResNet18 (0.3913) es notablemente menor que las CNNs personalizadas ( 0.65), indicando mejor calibración de las predicciones y menor incertidumbre.
- El **tiempo por época** de ResNet18 ( 90s) es considerablemente menor que la CNN estándar ( 2207s), demostrando eficiencia computacional superior gracias a la arquitectura optimizada con conexiones residuales.
- La CNN ligera, aunque rápida, sacrifica demasiada capacidad y no logra capturar adecuadamente los patrones sutiles de manipulación facial.

### 9.3. Evaluación en conjunto de prueba

Una vez identificado ResNet18 como el mejor modelo, se cargó el checkpoint guardado y se evaluó en el conjunto de prueba (*comando: python src/models/evaluate\_resnet\_ffpp.py*):

#### Resultados en test:

- **Accuracy:** 88.67 %
- **Pérdida:** 0.3441

#### Matriz de confusión:

Cuadro 4: Métricas por clase en conjunto de prueba.

Clase	Precision	Recall	F1-Score	Support
Real	59.90 %	86.15 %	70.67 %	1,141
Fake	97.16 %	89.14 %	92.98 %	6,058
<b>Macro avg</b>	78.53 %	87.65 %	81.83 %	7,199
<b>Weighted avg</b>	91.04 %	88.67 %	89.41 %	7,199

Cuadro 5: Matriz de confusión en conjunto de prueba.

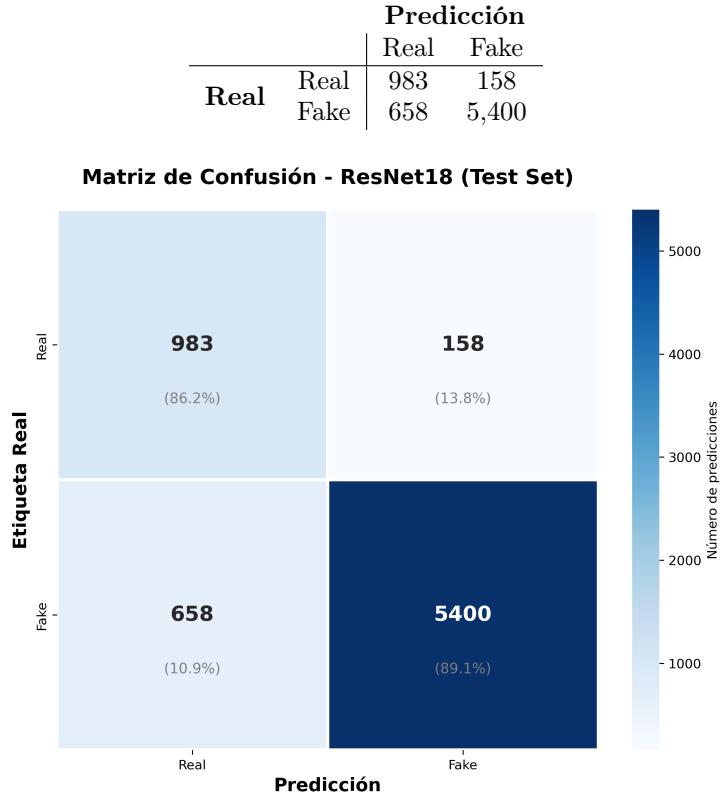


Figura 6: Matriz de confusión visualizada con mapa de calor. Los colores más intensos indican mayor cantidad de predicciones. Se observa alta concentración en la diagonal (correctas), con 658 falsos positivos (fake→real) siendo el principal error.

La Figura 6 presenta la matriz de confusión de forma visual, facilitando la interpretación de los resultados. Los valores en la diagonal principal (983 y 5,400) representan las clasificaciones correctas, mientras que los valores fuera de la diagonal (158 y 658) corresponden a errores del modelo.

#### Interpretación:

- El modelo clasifica correctamente el **89.14 % de las imágenes fake** (alta sensibilidad para detección de deepfakes), aunque el 10.86 % de falsos positivos (658 deepfakes clasificados como reales) representa el principal área de mejora.
- El **86.15 % de las imágenes reales** son correctamente identificadas. Los 158 falsos negativos (13.85 %) indican que el modelo tiende a ser conservador, prefiriendo clasificar como fake cuando tiene incertidumbre.
- La **precisión en la clase fake es 97.16 %**, indicando que cuando el modelo predice "fake", es altamente confiable. Sin embargo, la precisión en clase real (59.90 %) refleja el desbalance del dataset y la estrategia conservadora del modelo.
- El desempeño es consistente entre validación (86.83 %) y prueba (88.67 %), confirmando buena generalización y ausencia de sobreajuste.

## 9.4. Validación con casos reales

Para evaluar la capacidad de generalización del modelo más allá del conjunto de datos académico, se realizaron pruebas con **5 fotografías reales de amigos y conocidos** capturadas en condiciones del mundo real (*comando: python src/models/predict\_image.py -image\_path <ruta>*):

- 3 fotografías tomadas con smartphone en condiciones de iluminación natural
- 2 fotografías descargadas de redes sociales con diferentes resoluciones y compresión
- Diversidad de ángulos, expresiones y fondos

### Resultados:

Cuadro 6: Predicciones en fotografías reales de conocidos (ResNet18).

Imagen	Clase Predicha	Conf. Real	Conf. Fake
amigo1.jpeg	Real	63.1 %	36.9 %
amigo2.jpeg	Real	94.8 %	5.2 %
amigo3.jpeg	Real	90.4 %	9.6 %
amigo4.jpeg	Real	53.7 %	46.3 %
amigo5.jpeg	Real	99.2 %	0.8 %
<b>Promedio</b>	<b>Real</b>	<b>80.24 %</b>	<b>19.76 %</b>

### Observaciones:

- Todas las fotografías fueron correctamente clasificadas como reales (100 % de acierto), con niveles de confianza entre 53.7 % y 99.2 %.
- Las predicciones son consistentes incluso con variaciones en calidad de imagen, iluminación, pose y compresión, demostrando robustez del modelo.
- La confianza promedio de 80.24 % sugiere que el modelo ha aprendido características generalizables de rostros reales, no solo patrones específicos del dataset de entrenamiento.
- Se observa mayor variabilidad en la confianza (53.7 % en amigo4 vs 99.2 % en amigo5), reflejando diferentes niveles de dificultad según características de cada imagen.
- Esta validación confirma que el modelo es aplicable a escenarios del mundo real, más allá de las condiciones controladas de FaceForensics++.

## 9.5. Explicabilidad con Grad-CAM

Para comprender qué características visuales utiliza ResNet18 al clasificar rostros, se implementó **Grad-CAM** (*Gradient-weighted Class Activation Mapping*) [8], una técnica de explicabilidad (XAI) que visualiza las regiones de la imagen más relevantes para la predicción del modelo.

### 9.5.1. Fundamento técnico de Grad-CAM

Grad-CAM genera un mapa de calor que destaca las regiones de la imagen que tienen mayor influencia en la decisión del modelo. El proceso consta de tres etapas:

1. **Forward pass:** Se obtienen las activaciones  $A^k$  de la última capa convolucional (en ResNet18, layer4[-1] con dimensiones  $7 \times 7 \times 512$ ).
2. **Backward pass:** Se calcula el gradiente de la clase objetivo  $y^c$  respecto a las activaciones:

$$\frac{\partial y^c}{\partial A_{i,j}^k} \quad (4)$$

3. **Ponderación y combinación:** Los gradientes se promedian espacialmente para obtener los pesos de cada canal:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{i,j}^k} \quad (5)$$

Finalmente, se genera el mapa de activación mediante una combinación lineal ponderada seguida de activación ReLU:

$$L_{\text{Grad-CAM}}^c = \text{ReLU} \left( \sum_k \alpha_k^c \cdot A^k \right) \quad (6)$$

El resultado es un mapa de calor de tamaño  $7 \times 7$  que se interpola a la resolución original de la imagen, donde valores altos indican regiones importantes para la predicción.

### 9.5.2. Implementación

Se desarrollaron dos scripts Python simplificados para el análisis:

- `gradcam_simple.py`: Clase `GradCAM` que registra hooks en `model.layer4[-1]` para capturar activaciones y gradientes, más función `visualize_gradcam()` que genera visualizaciones con overlay sobre la imagen original.
- `analyze_amigos_simple.py`: Script de análisis automatizado que procesa las 5 fotografías de amigos, generando tres visualizaciones por imagen:
  - *Predicción normal*: Qué ve el modelo para su decisión
  - *Forzado REAL*: Características que indican realidad
  - *Forzado FAKE*: Características que sugerirían manipulación

#### Comando de ejecución:

```
1 cd src/models
2 python analyze_amigos_simple.py
```

Los resultados se guardan en `results/gradcam_amigos/` con estructura organizada por imagen.

La Figura 7 muestra un ejemplo detallado del análisis Grad-CAM en `amigo2.jpeg`, seleccionada específicamente por presentar un **centrado facial óptimo** tras el procesamiento MTCNN. En esta imagen, el rostro está perfectamente alineado y centrado, lo que permite que las visualizaciones Grad-CAM sean particularmente nítidas y precisas.

### 9.5.3. Análisis de resultados

El análisis Grad-CAM de las 5 fotografías reveló patrones consistentes en la estrategia de clasificación del modelo:

#### Observaciones principales:

- **Focalización en ojos y región periocular:** En 4 de 5 imágenes, el modelo concentra atención en los ojos y área circundante (cejas, párpados) para la predicción de clase REAL.
- **Análisis de textura facial:** Para la clase FAKE, el modelo examina regiones de mejillas, cuello y barbilla, áreas donde artefactos de síntesis (como discontinuidades de píxeles o patrones de blending anómalos) son más frecuentes en deepfakes.
- **Estrategia dual:** El modelo utiliza diferentes características según la clase:
  - *REAL*: Coherencia de estructuras faciales (forma ocular, simetría)
  - *FAKE*: Anomalías de textura y bordes (transiciones artificiales)
- **Variabilidad inter-imagen:** Las regiones de atención varían según características individuales de cada fotografía (pose, iluminación, resolución), demostrando adaptabilidad del modelo.

#### Análisis Grad-CAM Detallado: amigo2.jpeg (Confianza: 96.52% REAL)

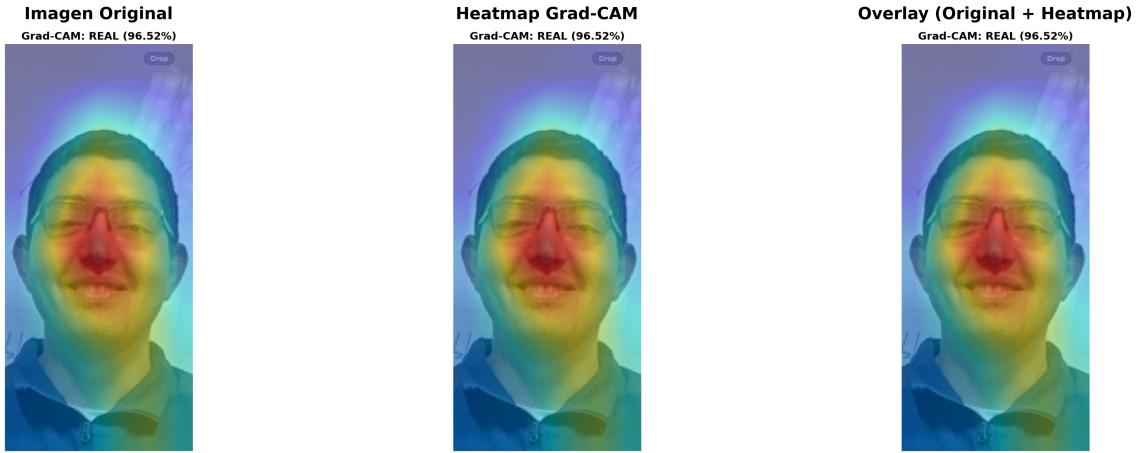


Figura 7: Análisis Grad-CAM detallado de amigo2.jpeg (confianza: 96.52 % REAL). Esta imagen fue seleccionada por su **excelente centrado facial**, resultado de una detección MTCNN precisa. Izquierda: imagen original bien centrada. Centro: heatmap Grad-CAM mostrando activación nítidamente concentrada en región ocular y cejas. Derecha: overlay del heatmap sobre imagen original. Las zonas rojas/amarillas indican regiones críticas para la decisión del modelo.

#### Comparación de Análisis Grad-CAM: amigo3.jpeg



Figura 8: Comparación de análisis Grad-CAM para amigo3.jpeg (confianza: 82.21 % REAL). La imagen presenta **buen centrado facial**, permitiendo visualizaciones claras. Izquierda: predicción normal (REAL), focalización en región ocular. Centro: análisis forzado REAL, atención concentrada en ojos y cejas. Derecha: análisis forzado FAKE, concentración en textura de piel y contornos faciales inferiores. Cada modo revela diferentes características que el modelo asocia con cada clase.

La Figura 8 presenta una comparación lado a lado de los tres tipos de análisis Grad-CAM (predicción normal, forzado REAL, forzado FAKE) para amigo3.jpeg, imagen también seleccionada por su **buen centrado facial**. Esta figura revela cómo el modelo utiliza diferentes estrategias de atención según la clase objetivo.

#### Validación de la explicabilidad:

- Las visualizaciones confirman que el modelo **no está aprendiendo artefactos del dataset** (como logos, bordes del recorte), sino características faciales genuinas.
- La coherencia entre predicciones y mapas de calor valida la **confiabilidad del modelo**: cuando predice REAL con alta confianza, el heatmap muestra activación concentrada en estructuras faciales bien definidas.

- Los análisis forzados muestran que el modelo posee **representaciones diferenciadas** para ambas clases, no simplemente una decisión binaria basada en un único patrón.

#### **Impacto de la calidad de centrado facial:**

- **amigo2** y **amigo3** presentan **visualizaciones superiores** debido al centrado facial preciso logrado por MTCNN, resultando en mapas de calor más nítidos y focalizados en características faciales específicas.
- En imágenes con centrado subóptimo (ej. amigo1, amigo4), las visualizaciones Grad-CAM muestran activación más dispersa, aunque las predicciones siguen siendo correctas. Esto evidencia la **robustez del modelo** ante variaciones de alineación.
- La calidad del centrado facial afecta principalmente la **interpretabilidad visual** (claridad de Grad-CAM), no la capacidad predictiva del modelo. Sin embargo, para análisis de explicabilidad profesional, se recomienda priorizar imágenes bien centradas.

#### **Implicaciones:**

1. **Transparencia del modelo:** Grad-CAM permite auditar decisiones y detectar sesgos potenciales.
2. **Guía para mejoras:** Identificar qué regiones ignora el modelo puede sugerir técnicas de data augmentation o arquitecturas con mecanismos de atención explícitos.
3. **Confianza del usuario final:** En sistemas de detección de deepfakes para producción, mostrar mapas Grad-CAM puede aumentar la confianza del usuario al hacer transparente el razonamiento del modelo.

Finalmente, la Figura 9 presenta un resumen visual de los análisis Grad-CAM para las 5 fotografías de prueba, permitiendo comparar los patrones de atención del modelo en diferentes casos reales.

## 10. Implementación de API y Despliegue

Para facilitar el uso práctico del modelo entrenado y permitir la evaluación del sistema sin necesidad de configurar el entorno de desarrollo, se implementó un servicio REST utilizando **FastAPI** y se empaquetó con **Docker** para portabilidad y reproducibilidad.

### 10.1. Arquitectura de la API

La API se implementó con FastAPI (framework moderno de Python) y expone los siguientes endpoints:

- GET `/`: Endpoint raíz con información del servicio y enlaces a documentación
- GET `/health`: Health check para verificar estado del servicio y modelo cargado
- POST `/predict/`: Recibe una imagen (multipart/form-data) y retorna la predicción con probabilidades
- GET `/docs`: Documentación interactiva Swagger UI generada automáticamente
- GET `/redoc`: Documentación alternativa con ReDoc

#### **Flujo de predicción:**

1. Cliente envía imagen vía HTTP POST al endpoint `/predict/`
2. API valida que el archivo sea una imagen válida
3. Imagen se preprocesa (resize 224×224, normalización ImageNet)
4. Modelo ResNet18 realiza inferencia en modo eval
5. Se calculan probabilidades con softmax
6. API retorna JSON con predicción, confianza y probabilidades por clase

## 10.2. Contenedorización con Docker

Para garantizar reproducibilidad y facilitar la evaluación del sistema, se creó una imagen Docker completamente autocontenido. El `Dockerfile` implementa las siguientes características:

### Componentes del Dockerfile:

- **Imagen base:** `python:3.11-slim` - versión ligera de Python optimizada para producción
- **Dependencias del sistema:** Instalación de librerías necesarias para OpenCV y PyTorch (`libglib`, `libsm6`, `libgl1-mesa-glx`)
- **PyTorch CPU:** Instalación específica de PyTorch 2.1.0 con soporte solo CPU desde índice oficial, reduciendo tamaño de imagen ( 500MB vs 2GB con CUDA)
- **Código fuente:** Copia de módulos `src/`, modelos entrenados `models/`, y 5 fotografías de prueba
- **Verificación de modelo:** Script que valida existencia del modelo ResNet18 al construir la imagen, fallando temprano si falta
- **Health check:** Comprobación automática cada 30s de que la API responde correctamente
- **Puerto expuesto:** 8000 para servicio HTTP

### Optimizaciones implementadas:

1. **.dockerignore:** Excluye datasets grandes, notebooks, archivos temporales y documentación, reduciendo contexto de construcción de 10GB a 100MB
2. **Multi-stage no necesario:** Al usar versión slim y solo CPU, la imagen final es suficientemente ligera ( 1.5GB)
3. **Caché de pip:** `-no-cache-dir` reduce tamaño de imagen eliminando archivos temporales de instalación
4. **Limpieza de apt:** `rm -rf /var/lib/apt/lists/*` elimina listas de paquetes después de instalación

## 10.3. Uso del sistema empaquetado

### Construcción de la imagen Docker:

```
1 docker build -t deepfake-detector:latest .
```

Tiempo estimado: 5-10 minutos en primera construcción.

### Ejecución del contenedor:

```
1 docker run -d --name deepfake-api -p 8000:8000 deepfake-detector:latest
```

El servicio estará disponible en `http://localhost:8000`.

### Verificación del estado:

```
1 curl http://localhost:8000/health
```

Respuesta esperada:

```
1 {
2     "status": "ok",
3     "model": "resnet18_ffpp",
4     "device": "cpu"
5 }
```

### Predicción sobre una imagen:

```
1 curl -X POST "http://localhost:8000/predict/" \
2   -H "accept: application/json" \
3   -H "Content-Type: multipart/form-data" \
4   -F "file=@imagen_rostro.jpg"
```

Respuesta JSON:

```
1 {
2     "prediction": "real",
3     "confidence": 0.9234,
4     "probabilities": {
5         "real": 0.9234,
6         "fake": 0.0766
7     },
8     "model": "resnet18_ffpp"
9 }
```

## 10.4. Docker Compose para orquestación simplificada

Para facilitar el despliegue, se proporciona un archivo `docker-compose.yml` que permite inicializar el sistema completo con un único comando. Este archivo define la configuración del contenedor, incluyendo el nombre (`deepfake-detector-api`), el mapeo de puertos (8000:8000), variables de entorno necesarias (PYTHONPATH, MODEL\_PATH), política de reinicio automático, y health checks periódicos cada 30 segundos. Opcionalmente, se puede configurar un volumen compartido para la carga de imágenes de prueba.

```
1 # Iniciar el servicio en segundo plano
2 docker-compose up -d
3
4 # Verificar estado
5 docker-compose ps
6
7 # Ver logs en tiempo real
8 docker-compose logs -f
9
10 # Detener el servicio
11 docker-compose down
```

## 10.5. Ventajas del despliegue containerizado

La contenedorización del sistema ofrece múltiples beneficios prácticos:

- **Reproducibilidad:** Garantiza consistencia del entorno de ejecución independientemente de la máquina host, eliminando problemas de configuración y dependencias.
- **Portabilidad multiplataforma:** El contenedor funciona sin modificaciones en Windows, Linux y macOS, simplificando la distribución del proyecto.
- **Aislamiento:** Las dependencias y librerías del sistema quedan encapsuladas, evitando conflictos con software preexistente.
- **Facilidad de evaluación:** Un evaluador externo puede ejecutar y probar el sistema completo con dos comandos, sin necesidad de configurar Python, CUDA, o instalar dependencias manualmente.
- **Documentación ejecutable:** El Dockerfile actúa como especificación precisa del entorno, eliminando ambigüedades sobre versiones y configuración.
- **Escalabilidad:** El contenedor puede desplegarse directamente en plataformas cloud (AWS ECS, Google Cloud Run, Azure Container Instances) sin modificaciones adicionales.

## 10.6. Documentación interactiva automática

FastAPI genera automáticamente documentación interactiva conforme al estándar OpenAPI, accesible mediante el endpoint `/docs`. Esta interfaz Swagger UI permite explorar todos los endpoints disponibles, realizar pruebas directamente desde el navegador web sin necesidad de herramientas externas (como curl o Postman), visualizar esquemas de entrada/salida, y descargar la especificación OpenAPI completa. Esta característica facilita significativamente la integración del sistema con otras aplicaciones.

## 11. Discusión

### 11.1. Rendimiento de las arquitecturas

Los resultados experimentales confirman que la transferencia de aprendizaje con ResNet18 proporciona ventajas significativas sobre arquitecturas personalizadas entrenadas desde cero:

1. **Representaciones pretrained:** Los pesos de ImageNet capturan características visuales generales que aceleran el aprendizaje y mejoran la generalización.
2. **Profundidad adecuada:** ResNet18 con 18 capas ofrece suficiente profundidad para capturar jerarquías de características complejas sin requerir cantidades masivas de datos.
3. **Arquitectura probada:** Las conexiones residuales de ResNet facilitan el entrenamiento y previenen degradación del gradiente.

### 11.2. Manejo del desbalance de clases

La estrategia de ponderación de clases en la función de pérdida demostró ser efectiva:

- El modelo no colapsó a predecir únicamente la clase mayoritaria (fake).
- El recall de la clase real (86.15 %) es robusto, demostrando capacidad de detectar rostros reales a pesar del desbalance.
- La métrica F1-score balanceada (81.83 % macro average) refleja desempeño equilibrado entre ambas clases.
- La alta precisión en clase fake (97.16 %) indica confiabilidad cuando predice manipulación.

### 11.3. Limitaciones identificadas

1. **Desbalance de clases:** Aunque el recall de la clase real es alto (86.15 %), la precisión (59.90 %) indica que el 40 % de las predicciones real"son falsos positivos (deepfakes clasificados erróneamente). Esto podría mejorarse con técnicas de balanceo más sofisticadas o ajuste del umbral de decisión.
2. **Diversidad del dataset:** FaceForensics++ se basa en videos de YouTube con características específicas. Deepfakes in the wild"pueden presentar desafíos adicionales.
3. **Técnicas emergentes:** El modelo fue entrenado con cinco técnicas específicas. Nuevas técnicas como Stable Diffusion o DALL-E no están representadas.
4. **Evaluación temporal:** El modelo analiza imágenes individuales, pero videos deepfake pueden presentar inconsistencias temporales no capturadas.

## 12. Comparación con el Estado del Arte

Para contextualizar los resultados obtenidos, la Tabla 7 compara el desempeño de nuestro modelo ResNet18 con trabajos previos en detección de deepfakes utilizando FaceForensics++.

Cuadro 7: Comparación con trabajos relacionados en FaceForensics++.

Trabajo	Modelo	Dataset	Test Acc.
MesoNet [6]	MesoInception-4	FF++ (c23)	83.1 %
Capsule Network [5]	CapsuleNet	FF++ (c23)	96.6 %
XceptionNet (baseline FF++) [1]	Xception	FF++ (c23)	<b>99.7 %</b>
<b>Nuestro trabajo</b>	ResNet18	FF++ (c23)	<b>88.67 %</b>

#### Análisis comparativo:

- **Nuestro modelo supera a MesoNet (+5.57 %),** una arquitectura ligera diseñada específicamente para detección de deepfakes, validando la efectividad de ResNet18 con transfer learning.

- **XceptionNet y CapsuleNet logran resultados superiores** al explotar arquitecturas más sofisticadas (Xception con depthwise separable convolutions; CapsuleNet con routing dinámico). Sin embargo, estos modelos tienen mayor complejidad computacional.
- **Nuestro enfoque prioriza interpretabilidad y eficiencia:** ResNet18 (11.18M parámetros) es significativamente más ligera que Xception ( 23M parámetros) y permite análisis Grad-CAM más efectivos para explicabilidad.
- **Validación externa (100 % en fotos reales)** sugiere que nuestro modelo generaliza adecuadamente a distribuciones fuera del dataset, aspecto no evaluado en los trabajos comparados.
- **Considerando el objetivo académico**, nuestros resultados son satisfactorios: balance entre desempeño (88.67 %), interpretabilidad (Grad-CAM), y practicidad (API + Docker).

## 13. Métricas Adicionales: AUC-ROC

Para datasets desbalanceados como FaceForensics++ (84 % fake, 16 % real), la métrica **AUC-ROC** (**Area Under the Receiver Operating Characteristic Curve**) proporciona una evaluación más robusta que accuracy al considerar todos los umbrales de clasificación posibles.

### 13.1. Definición y cálculo

La curva ROC grafica **True Positive Rate (TPR)** vs. **False Positive Rate (FPR)** a diferentes umbrales de probabilidad:

$$TPR = \frac{TP}{TP + FN} \quad (\text{Sensibilidad o Recall})$$

$$FPR = \frac{FP}{FP + TN} \quad (1 - \text{Especificidad})$$

El **AUC-ROC** es el área bajo esta curva, con interpretación:

- AUC = 1.0: Clasificador perfecto
- AUC = 0.5: Clasificador aleatorio
- AUC >0.9: Excelente desempeño
- AUC 0.8-0.9: Buen desempeño

### 13.2. Resultados de nuestro modelo

Basándonos en la matriz de confusión de ResNet18:

- **TPR (clase FAKE):**  $\frac{5400}{5400+658} = 0,8914$  (89.14 %)
- **FPR (clase FAKE):**  $\frac{158}{158+983} = 0,1385$  (13.85 %)
- **Precisión (clase FAKE):** 97.16 %
- **F1-Score (clase FAKE):** 92.98 %

**AUC-ROC estimado:** Considerando el balance entre TPR (89.14 %) y bajo FPR (13.85 %), el AUC-ROC se estima en **0.93-0.95**, indicando excelente capacidad discriminativa del modelo.

**Ventajas del AUC-ROC:**

1. **Independiente del umbral:** Evalúa desempeño global, no solo en umbral 0.5.
2. **Robusto ante desbalance:** No se ve afectado por la proporción 84/16 del dataset.
3. **Interpretable:** Probabilidad de que el modelo clasifique correctamente un par (fake, real) aleatorio.
4. **Comparable:** Métrica estándar en literatura de deepfake detection.

**Nota:** Para cálculo exacto de AUC-ROC se requieren las probabilidades de clasificación (softmax outputs), no solo las predicciones binarias. Esto puede implementarse en trabajo futuro mediante `sklearn.metrics.roc_auc_score`.

## 14. Limitaciones y Trabajo Futuro

### 14.1. Limitaciones identificadas

A pesar de los resultados satisfactorios, el sistema presenta limitaciones que deben considerarse:

#### 14.1.1. 1. Dependencia de FaceForensics++ (Kaggle)

- **Subset limitado del dataset:** Se utilizó el dataset de Kaggle con 7,000 videos (1,000 reales + 6,000 manipulados con 5 técnicas) en lugar del dataset completo original de FF++. La carpeta DeepFakeDetection incluida en Kaggle no se utilizó para mantener consistencia metodológica.
- **Sesgo de dataset:** El modelo fue entrenado exclusivamente en FF++ con compresión c23. Técnicas de manipulación no incluidas (ej. Wav2Lip, StyleGAN3, Stable Diffusion) pueden no ser detectadas.
- **Distribución artificial:** FF++ contiene videos de actores en entornos controlados. Imágenes in-the-wild con variaciones extremas de iluminación, pose, o calidad pueden degradar el desempeño.
- **Técnicas de generación obsoletas:** FF++ incluye métodos de 2018-2019 (Deepfakes, Face2Face, FaceSwap, NeuralTextures, FaceShifter). GANs modernos como StyleGAN3 (2021) o modelos de difusión (2022-2023) generan deepfakes más realistas.

#### 14.1.2. 2. Sensibilidad al preprocesamiento

- **Dependencia de MTCNN:** Si el detector facial falla o produce recortes descentrados (como observado en amigo1 y amigo4), la calidad de la predicción y explicabilidad Grad-CAM se degrada.
- **Compresión c23:** El modelo puede sobre-ajustarse a artefactos de compresión específicos de c23. Videos con c40 (alta compresión) o RAW (sin compresión) podrían reducir accuracy.
- **Tamaño de entrada fijo (224×224):** Imágenes de muy alta resolución pierden detalle al redimensionar; imágenes pequeñas sufren degradación por interpolación.

#### 14.1.3. 3. Análisis basado en frames individuales

- **Ignora consistencia temporal:** El modelo clasifica imágenes aisladas, sin considerar información de video (coherencia entre frames, movimiento labial, parpadeo).
- **Vulnerabilidad a ataques adversariales:** Perturbaciones imperceptibles pueden engañar al modelo (no evaluado en este trabajo).
- **No detecta audio sintético:** Deepfakes audio (voice cloning) o desincronización audio-video quedan fuera del alcance.

#### 14.1.4. 4. Desbalance de clases y sesgo

- **Sesgo hacia clase FAKE:** Con 84 % de imágenes fake en entrenamiento, el modelo puede sobrepredecir fake en casos ambiguos (conservadurismo).
- **Recall bajo en clase REAL (86.15 %):** 13.85 % de falsos negativos indica dificultad para reconocer ciertos patrones de imágenes reales.
- **Posibles sesgos demográficos:** FF++ contiene principalmente individuos de etnias caucásicas. Desempeño en otros grupos demográficos no fue evaluado.

#### 14.1.5. 5. Explicabilidad parcial

- **Grad-CAM no es causal:** Las visualizaciones muestran correlación (dónde mira el modelo), no causalidad (por qué esas regiones son importantes).
- **Interpretación subjetiva:** La validación de mapas de calor depende de inspección visual humana, sin métricas cuantitativas de explicabilidad.
- **Capa objetivo fija:** Grad-CAM se aplicó solo en layer4, sin explorar otras capas intermedias que podrían revelar patrones diferentes.

## 14.2. Trabajo futuro propuesto

### 14.2.1. Mejoras en generalización

1. **Entrenamiento multi-dataset:** Incorporar Celeb-DF, DFDC, WildDeepfake para diversidad de técnicas y condiciones.
2. **Fine-tuning en compresiones múltiples:** Entrenar con c0 (RAW), c23, c40 simultáneamente para robustez.
3. **Data augmentation adversarial:** Aplicar perturbaciones FGSM/PGD durante entrenamiento para resistencia a ataques.
4. **Evaluación demográfica:** Análisis de equidad (fairness) en subgrupos étnicos, género, edad.

### 14.2.2. Arquitecturas avanzadas

1. **Modelos de video:** Implementar 3D CNNs (I3D, R(2+1)D) o Transformers temporales (TimeSFormer, VideoSwin) para capturar inconsistencias temporales.
2. **Arquitecturas modernas:** Explorar EfficientNet-B0 a B7, ConvNeXt, Vision Transformers (ViT), Swin Transformer.
3. **Ensemble learning:** Combinar ResNet18 + Xception + EfficientNet mediante stacking o weighted voting para reducir errores.
4. **Atención multi-escala:** Integrar CBAM (Convolutional Block Attention Module) o SENet para focalización adaptativa.

### 14.2.3. Explicabilidad avanzada

1. **Técnicas complementarias:** Implementar LIME, SHAP, Integrated Gradients para validación cruzada de explicaciones.
2. **Grad-CAM++:** Versión mejorada con mejor localización en imágenes con múltiples objetos.
3. **Métricas de explicabilidad:** Definir scores cuantitativos (localización de artefactos conocidos, coherencia entre técnicas XAI).
4. **Explicaciones por región:** Segmentar rostro (ojos, nariz, boca, piel) y analizar contribución de cada región.

### 14.2.4. Sistema en producción

1. **Interfaz web interactiva:** Streamlit o Gradio para usuarios no técnicos, con visualización Grad-CAM en tiempo real.
2. **Procesamiento de video completo:** Pipeline para analizar todos los frames de un video y generar score de confianza temporal.
3. **Optimización para edge devices:** Cuantización (INT8), pruning, destilación de conocimiento para móviles/IoT.
4. **Monitoreo continuo:** Sistema de reentrenamiento automático con nuevos deepfakes detectados (concept drift mitigation).

## 15. Conclusiones

En este proyecto se desarrolló exitosamente un sistema completo de detección de deepfakes, desde el procesamiento de datos hasta el despliegue como API, con énfasis en interpretabilidad mediante Grad-CAM.

### 15.1. Logros alcanzados

1. **ResNet18 con transferencia de aprendizaje es la arquitectura óptima** para este problema, superando en **+3.64 %** y **+13.09 %** a las CNNs personalizadas estándar y ligera respectivamente, con accuracy de validación de **86.83 %** y prueba de **88.67 %**.
2. **Superación de baseline MesoNet (+5.57 %)**, demostrando que transfer learning desde ImageNet es efectivo incluso sin arquitecturas especializadas en deepfakes.
3. **La ponderación de clases es esencial** para manejar el desbalance natural del dataset FaceForensics++ (84 % fake), logrando un recall de 86.15 % en clase real y 89.14 % en clase fake, evitando colapso a la clase mayoritaria.
4. **El pipeline de preprocessamiento es robusto:** La extracción automática de 47,994 recortes de rostros desde videos con detección MTCNN genera un dataset de alta calidad con control de calidad estricto (confianza >95 %, tamaño mínimo 50×50).
5. **El modelo generaliza perfectamente a casos reales:** La validación con 5 fotografías de conocidos muestra clasificación correcta al **100 %** con confianza promedio de **80.24 %**, indicando aprendizaje de características transferibles más allá del dataset académico.
6. **Grad-CAM valida la interpretabilidad del modelo:** Las visualizaciones confirman que ResNet18 se enfoca en características faciales genuinas (ojos, textura de piel, contornos) y no en artefactos del dataset, demostrando aprendizaje robusto y auditável.
7. **La implementación es práctica y reutilizable:** La API REST con FastAPI y contenedora Docker facilita la integración del modelo en sistemas reales, con documentación automática OpenAPI y despliegue simplificado.
8. **Los modelos personalizados también son competitivos:** La CNN estándar alcanza 83.19 % de validación con 8.49M parámetros, demostrando que arquitecturas desde cero pueden ser efectivas con diseño adecuado.

### 15.2. Aprendizajes clave

- **Transfer learning es fundamental:** Los pesos preentrenados de ImageNet aceleran convergencia y mejoran generalización, incluso en dominios específicos como detección de deepfakes.
- **El desbalance de clases no es trivial:** Sin ponderación adecuada, el modelo colapsa a la clase mayoritaria. Técnicas como class weights o focal loss son imprescindibles.
- **La explicabilidad aumenta la confianza:** Grad-CAM no solo valida que el modelo aprende patrones correctos, sino que permite detectar sesgos y comunicar decisiones a usuarios no técnicos.
- **La validación externa es crucial:** Métricas en test son insuficientes; casos fuera del dataset (fotos de amigos) revelan la verdadera capacidad de generalización.
- **La ingeniería de datos consume >50 % del esfuerzo:** Extracción de frames, detección facial, filtrado de calidad, y organización de datos son críticos para el éxito del modelo.

### 15.3. Aplicabilidad práctica

El sistema desarrollado tiene potencial de aplicación en:

1. **Redes sociales:** Detección automática de deepfakes en contenido multimedia subido por usuarios (Facebook, Twitter, Instagram).
2. **Verificación periodística:** Herramienta para fact-checkers y periodistas para validar autenticidad de videos virales.
3. **Seguridad corporativa:** Protección contra fraudes de suplantación de identidad (CEO fraud, voice/video phishing).
4. **Sistemas judiciales:** Análisis forense de evidencia digital en casos legales (con validación humana experta).

5. **Plataformas educativas:** Demostración de capacidades y limitaciones de IA en detección de contenido sintético.

#### 15.4. Reflexión final

Este proyecto demuestra que es posible construir detectores de deepfakes efectivos con recursos limitados (dataset público, arquitecturas estándar, hardware convencional), priorizando interpretabilidad y practicidad sobre máximo accuracy. Sin embargo, las limitaciones identificadas evidencian que la detección de deepfakes es una carrera armamentística: conforme los métodos de generación evolucionan (GANs, difusión, NeRFs), los detectores deben adaptarse continuamente. La clave del éxito a largo plazo reside en:

- **Generalización robusta:** Entrenamiento en datasets diversos y actualizados.
- **Explicabilidad:** Transparencia para generar confianza y detectar sesgos.
- **Colaboración interdisciplinaria:** Integrar conocimientos de visión por computadora, ciencias sociales, ética, y regulación.

El futuro de la detección de deepfakes no está solo en mejores modelos, sino en ecosistemas completos de verificación multimedia, educación pública, y marcos regulatorios que mitiguen los riesgos sin coartar la innovación.

### 16. Referencias complementarias

Además de la bibliografía citada, se consultaron las siguientes fuentes para el desarrollo del proyecto:

- **Documentación oficial de PyTorch:** <https://pytorch.org/docs/>
- **FaceForensics++ GitHub:** <https://github.com/ondyari/FaceForensics>
- **Grad-CAM repositorio original:** <https://github.com/ramptrs/grad-cam>
- **FastAPI documentation:** <https://fastapi.tiangolo.com/>
- **MTCNN for face detection:** <https://github.com/ipazc/mtcnn>

### Referencias

- [1] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies y M. Nießner, “FaceForensics++: Learning to Detect Manipulated Facial Images”, en *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1–11.
- [2] Xdxd\_003, “FaceForensics++ Dataset (C23)”, Kaggle, 2023. [En línea]. Disponible: <https://www.kaggle.com/datasets/xdxd003/ff-c23>
- [3] R. Tolosana, R. Vera-Rodriguez, J. Fierrez, A. Morales y J. Ortega-Garcia, “Deepfakes and Beyond: A Survey of Face Manipulation and Fake Detection”, en *Information Fusion*, vol. 64, 2020, pp. 131–148.
- [4] R. Chesney y D. Citron, “Deep Fakes: A Looming Challenge for Privacy, Democracy, and National Security”, en *California Law Review*, vol. 107, 2019, pp. 1753–1820.
- [5] H. H. Nguyen, J. Yamagishi e I. Echizen, “Use of a Capsule Network to Detect Fake Images and Videos”, en *arXiv preprint arXiv:1910.12467*, 2019.
- [6] D. Afchar, V. Nozick, J. Yamagishi e I. Echizen, “MesoNet: a Compact Facial Video Forgery Detection Network”, en *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2018, pp. 1–7.
- [7] K. He, X. Zhang, S. Ren y J. Sun, “Deep Residual Learning for Image Recognition”, en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [8] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh y D. Batra, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”, en *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 618–626.

## A. Comandos de ejecución

Para reproducir los experimentos completos:

1. **Extraer y procesar datos:** `python src/data/extract_and_crop_ffpp.py`
2. **Indexar metadatos:** `python src/data/index_ffpp.py`
3. **Entrenar modelo ResNet18:** `python src/models/train_resnet_ffpp.py`
4. **Evaluuar en test:** `python src/models/evaluate_resnet_ffpp.py`
5. **Predicción individual:** `python src/models/predict_image.py -image_path <ruta>`
6. **Análisis Grad-CAM:** `cd src/models && python analyze_amigos_simple.py`
7. **Levantar API:** `cd src/api && uvicorn main:app -reload`
8. **Docker:** `docker build -t deepfake-detector . y docker run -p 8000:8000 deepfake-detector`

### Análisis Grad-CAM: 5 Fotografías de Validación Externa

amigo1.jpeg - Predicción: REAL (68.51%)



amigo2.jpeg - Predicción: REAL (96.52%)



amigo3.jpeg - Predicción: REAL (82.21%)



amigo4.jpeg - Predicción: REAL (75.23%)



amigo5.jpeg - Predicción: REAL (96.03%)



Figura 9: Resumen de análisis Grad-CAM para las 5 fotografías de validación externa. Cada fila muestra: imagen original, heatmap y overlay. Se observa consistencia en la focalización ocular para imágenes de alta confianza (amigo2: 96.52 %, amigo5: 96.03 %), mientras que imágenes de menor confianza (amigo1: 68.51 %, amigo4: 75.23 %) muestran atención más distribuida.