Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Querétaro

Escuela de ingeniería y ciencias

Desarrollo de aplicaciones avanzadas de ciencias computacionales

Grupo 201

Actividad 2 ML

Profesor:

Benjamín Valdés Aguirre

Integrante:

Juan Pablo Cabrera Quiroga | A01661090

Fecha de entrega:

29 de Abril del 2025

Índice.

Introducción.

En el internet del mundo actual, la cantidad de noticias falsas y, en particular, de contenido visual generado por AI, representa un desafío significativo para la sociedad. Cada vez es más difícil discernir entre lo que es real y lo que es virtualmente generado. Esto derivado de que las herramientas para crear este tipo de contenido están al alcance de cualquier persona, incluso sin conocimientos técnicos avanzados. Esta problemática se vuelve aún más importante en contextos políticos o cuando involucra a figuras públicas, ya que la manipulación de información visual puede influir negativamente en la percepción social y generar consecuencias serias, entre muchos otros de los efectos negativos que puede llegar a tener.

Este proyecto aborda el reto de la detección de imágenes falsas mediante el desarrollo de un sistema basado en Redes Neuronales Convolucionales (CNN). Siguiendo el estado del arte establecido por Kroiß y Reschke en su trabajo "Deepfake Detection of Face Images based on a Convolutional Neural Network"[1], implementé un enfoque de clasificación binaria para distinguir entre imágenes reales y generadas artificialmente.

Mi modelo, al igual que el referenciado en el trabajo mencionado, utiliza una arquitectura ResNet-50 pre entrenada como base, aprovechando su eficacia comprobada en la clasificación de imágenes. Mediante técnicas de transfer learning y fine-tuning, adapté esta arquitectura al problema específico de detección de imágenes generadas por AI, agregando capas personalizadas para la clasificación binaria.

A diferencia del estudio original, que reportó métricas excelentes (precisión de 0.98, recall de 0.96, F1-Score de 0.97 y AUC de 0.99), mi modelo alcanzó resultados más modestos pero igualmente válidos, con un accuracy de 77.3%, precisión de 0.843, recall de 0.671 y F1-Score de 0.747. Esta diferencia podría atribuirse a variaciones en los conjuntos de datos utilizados y en los métodos específicos de preprocesamiento y entrenamiento.

En las siguientes secciones, detallaré la arquitectura del modelo, el proceso de preprocesamiento de datos, las estrategias de entrenamiento implementadas y los resultados obtenidos y posibles áreas de mejora para futuras iteraciones del sistema.

Pre-procesamiento.

Para el proceso de pre-procesamiento de mi modelo lleve a cabo una serie de pasos para asegurarse que mi modelo funciona de la mejor manera.

Verificación de Integridad de Imágenes

Antes de iniciar el procesamiento, realicé una verificación exhaustiva de las imágenes para identificar y gestionar archivos corruptos. Hice esto porque en la primera iteración de entrenamiento, me dió múltiples errores en relación a la integridad de las imágenes. Esta operación preventiva la implementé en el script `check_corrupted_images.py`, el cual:

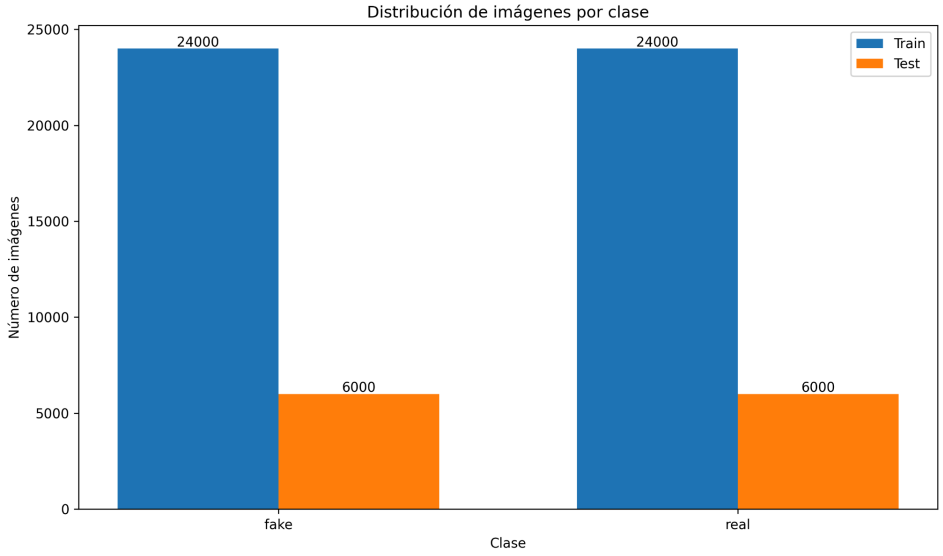
- Examina sistemáticamente todos los archivos en los directorios de entrenamiento y prueba
- Identifica imágenes corruptas o ilegibles
- Proporciona la opción de eliminar automáticamente los archivos problemáticos

Este paso fue fundamental para evitar que me volvieran a salir errores durante el entrenamiento y asegurar la calidad del conjunto de datos.

Normalización y Redimensionamiento

Para estandarizar el conjunto de datos, apliqué las siguientes transformaciones:

- **Redimensionamiento**: Todas las imágenes fueron ajustadas a una dimensión uniforme de 224×224 píxeles, un estándar compatible con la arquitectura ResNet50.
- **Normalización**: Los valores de píxeles fueron escalados al rango [0,1] mediante división por 255, facilitando la convergencia durante el entrenamiento



Técnicas de Data Augmentation

Para incrementar la robustez del modelo frente a variaciones en las imágenes y mitigar el riesgo de overfitting, implementé diversas técnicas de data augmentation:

- **Rotación aleatoria**: Aplicación de rotaciones aleatorias de hasta 20 grados.
- **Desplazamiento horizontal y vertical**: Movimientos aleatorios de hasta un 20% en dirección derecha e izquierda.
- **Volteo horizontal**: Voltee las imágenes en el eje horizontal

Estas transformaciones, configuradas en el módulo *`ImageDataGenerator`* de TensorFlow, generan variantes artificiales de las imágenes originales durante el entrenamiento, enriqueciendo efectivamente el conjunto de datos sin requerir imágenes adicionales.

Segmentación de Datos

Organice el conjunto de datos para optimizar el proceso de aprendizaje:

- **División training/validation**: Implementación de una división interna 80/20 del conjunto de entrenamiento para validación cruzada durante el entrenamiento
- **Conjunto de test independiente**: Mantenimiento de un conjunto separado para evaluación imparcial del rendimiento final

Generación Dinámica

Cómo trabajar con un dataset tan grande para mi representó hasta cierto punto un problema por mi capacidad computacional, implementé un sistema de generación dinámica de batches con las siguientes características:

- **Tamaño de lote**: Configuración de 32 imágenes por batch.
- **Mezcla aleatoria**: Reordenamiento de imágenes entre épocas para evitar patrones de aprendizaje relacionados con el orden
- **Procesamiento en tiempo real**: Aplicación de transformaciones durante el entrenamiento

  El haber hecho este preprocesamiento contribuyó significativamente a la capacidad de mi modelo para generalizar a partir de los patrones en las imágenes, distinguiendo a muy buen nivel entre contenido real y generado artificialmente.

Modelo y entrenamiento.

Para el modelo implementé una arquitectura basada en transfer learning utilizando ResNet50 como modelo base, siguiendo el enfoque planteado en el estado del arte. Las principales características de nuestra arquitectura son:

- **Backbone**: ResNet50 pre entrenado en ImageNet, aprovechando su capacidad para extraer características visuales complejas.
- **Fine-tuning selectivo**: Congelamiento de las primeras capas del modelo base, permitiendo el entrenamiento de solo las últimas 15 capas para adaptación específica. Esto también lo ví en el estado del arte.
- **Capas adicionales**:
    - Dos capas densas (512 y 256 neuronas) con batch normalization
    - Dropout agresivo (0.6) para mitigar overfitting
    - Capa de salida con activación sigmoide para clasificación binaria

En total, el modelo cuenta con 24,770,689 parámetros, de los cuales solo 6,701,825 (27%) son entrenables, lo que optimiza el proceso de entrenamiento y reduce el riesgo de sobreajuste.

Proceso de Entrenamiento

El entrenamiento de mi modelo se ejecutó con las siguientes configuraciones:

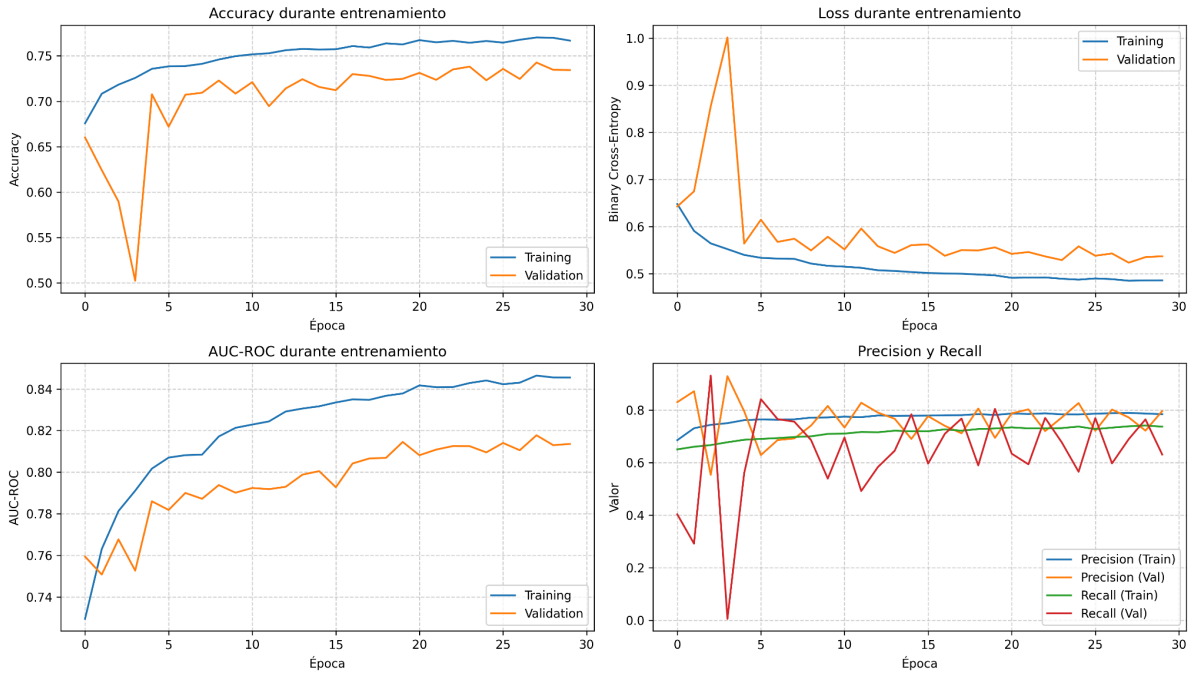
- **Optimizador**: Adam con learning rate inicial de 3e-4 (esto conforme iban pasando las épocas y alguno de los parámetros no mejoraba, como el accuracy o el loss, iba determinando que fuera dinámico el learning rate).
- **Función de pérdida**: Binary Cross-Entropy
- **Batch size**: 32 imágenes
- **Métricas de monitoreo**: Accuracy, AUC-ROC, Precision, Recall

Para mejorar la generalización y eficiencia, implementamos varias estrategias:

- **Early stopping**: Monitorización de la pérdida en validación con paciencia de 12 épocas. Aunque este nunca se activó completando el total de las épocas de entrenamiento.
- **Reducción dinámica del learning rate**: Factor de reducción de 0.5 cuando el aprendizaje se estancaba
- **Checkpoint de modelos**: Guardado del mejor modelo según accuracy en validación

Curvas de Aprendizaje

El entrenamiento duró 30 épocas, con una convergencia gradual. A continuación se muestran las curvas de aprendizaje que ilustran la evolución del modelo:



La gráfica muestra una mejora progresiva tanto en accuracy como en pérdida, con una brecha relativamente pequeña entre entrenamiento y validación, indicando un buen balance sin signos claros de overfitting. El learning rate adaptativo se redujo progresivamente desde 3e-4 hasta 9.4e-6, facilitando un ajuste fino en las últimas épocas.

El proceso completo tomó 28 horas de entrenamiento utilizando aceleración GPU, logrando una convergencia satisfactoria con el modelo final alcanzando su mejor rendimiento en las épocas finales.

Evaluación y análisis de resultados.

Modelo Inicial

Mi primer modelo implementado consistió en una arquitectura de transfer learning básica con ResNet50 como backbone, siguiendo las recomendaciones generales del paper del estado del arte. Este modelo inicial presentaba las siguientes características:

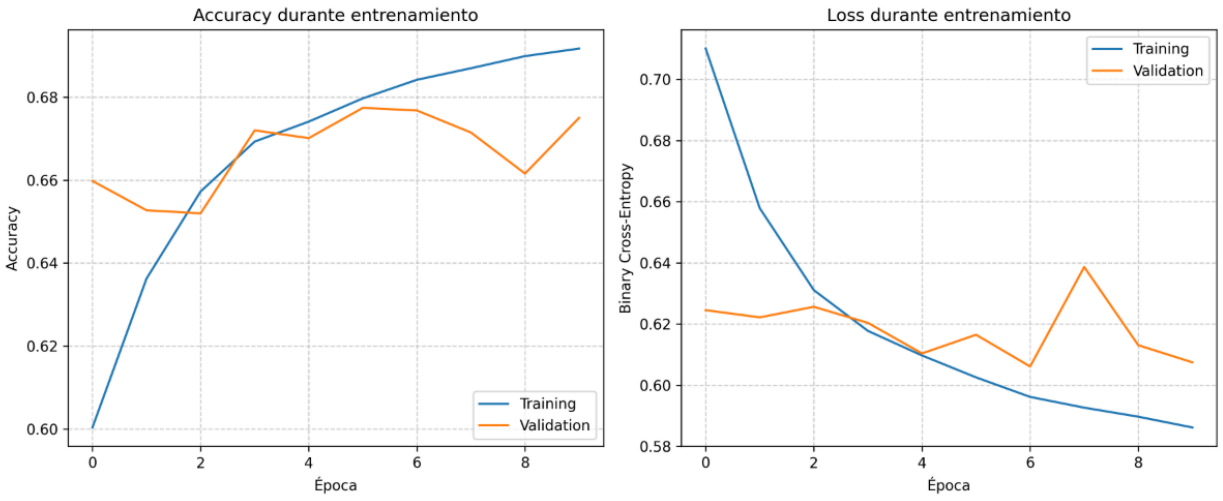
- **Backbone**: ResNet50 pre entrenado en ImageNet con todas las capas congeladas
- **Capas adicionales**: Una capa densa de 512 neuronas con activación ReLU
- **Regularización**: Dropout de 0.5 para mitigar overfitting
- **Capa de salida**: Una neurona con activación sigmoide para clasificación binaria
- **Optimizador**: Adam con learning rate inicial de 1e-4
- **Entrenamiento**: 10 épocas con batch size de 32

Métricas Obtenidas

Tras el entrenamiento inicial, estas fueron las métricas de mi modelo en el conjunto de test:

Métrica	Valor
Accuracy	67.5%
AUC-ROC	0.740
Precision	0.693
Recall	0.627
F1-Score	0.659

Análisis de Curvas de Aprendizaje



El análisis de las curvas de aprendizaje de mi modelo inicial revela varios aspectos importantes:

1. **Convergencia limitada**: La accuracy de entrenamiento alcanzó aproximadamente un 69%, mientras que la de validation se estabilizó alrededor del 67.5%.
2. **Inestabilidad en validación**: Hay cambios significativos en la curva de validation, particularmente en las épocas 2-4, donde la accuracy descendió hasta aproximadamente el 50%, sugiriendo dificultades de generalización.
3. **Brecha entre entrenamiento y validación**: Aunque no hay señales claras de overfitting, mi modelo no logró capturar patrones complejos para distinguir eficazmente entre imágenes reales y falsas.

4. **Estancamiento temprano**: El progreso del modelo se estancó rápidamente, sin mejoras significativas después de la época 7, sugiriendo que mi modelo alcanzó su capacidad máxima con la arquitectura dada.

Limitaciones Identificadas

A partir del análisis del rendimiento de mi modelo inicial, se identificaron las siguientes limitaciones:

1. **Arquitectura insuficiente**: La arquitectura simplificada con una sola capa densa de 512 neuronas podría ser insuficiente para capturar las sutilezas que distinguen las imágenes falsas de las reales.
2. **Estrategia de regularización subóptima**: Aunque apliqué dropout, no utilicé técnicas como batch normalization que podrían estabilizar el entrenamiento.
3. **Duración de entrenamiento insuficiente**: Las 10 épocas podrían no ser suficientes para permitir que mi modelo converja adecuadamente, especialmente considerando el tamaño del conjunto de datos (48,000 imágenes de entrenamiento).
4. **Estrategia de optimización limitada**: La ausencia de un esquema de reducción progresiva del learning rate más agresivo podría haber impedido un fine tuning del modelo.

Estas limitaciones sirvieron como base para la propuesta de mejoras implementadas en la versión subsiguiente del modelo.

Propuesta de mejora.

Tomando en cuenta las limitaciones identificadas en mi modelo inicial y tomando como referencia las recomendaciones del paper de Kroiß y Reschke, hice las siguientes mejoras a mi modelo:

1. Arquitectura Mejorada

Fine-tuning selectivo: En lugar de congelar todo el backbone, hago que el entrenamiento de las capas superiores de ResNet50 se pueda adaptar a características de alto nivel al problema específico.

Profundidad arquitectónica: Incorporar múltiples capas densas con tamaños decrecientes (512→256→1) para permitir una abstracción progresiva de características.
Batch normalization: Implementé batch normalization antes de las activaciones para estabilizar el entrenamiento y acelerar la convergencia.

2. Estrategias de Regularización Avanzadas

Dropout agresivo: Aumentar la tasa de dropout de 0.5 a 0.6 para combatir más eficazmente el overfitting, especialmente importante al descongelar parte del backbone.

Más data augmentation: Incrementar el rango de transformaciones en data augmentation, ampliando la rotación de 15° a 20° y el desplazamiento de 0.1 a 0.2.

3. Optimización del Proceso de Entrenamiento

Duración extendida: Aumentar el número de épocas máximas de 10 a 30 para permitir una convergencia más completa.

Estrategia adaptativa de learning rate: Implementé un esquema más agresivo de reducción de learning rate, con factor de reducción de 0.5 y monitoreo más frecuente.

4. Monitoreo y Evaluación Integral

Visualización mejorada: Implementé visualización de métricas adicionales como AUC-ROC y Precision/Recall durante el entrenamiento.

Análisis posterior: Guardé información detallada sobre la matriz de confusión y métricas específicas por clase para un análisis más profundo.

Mejoras implementadas.

Tras implementar las mejoras propuestas, mi modelo actualizado fue entrenado durante 30 épocas. A continuación, estos son los resultados obtenidos y una comparación con el modelo inicial.

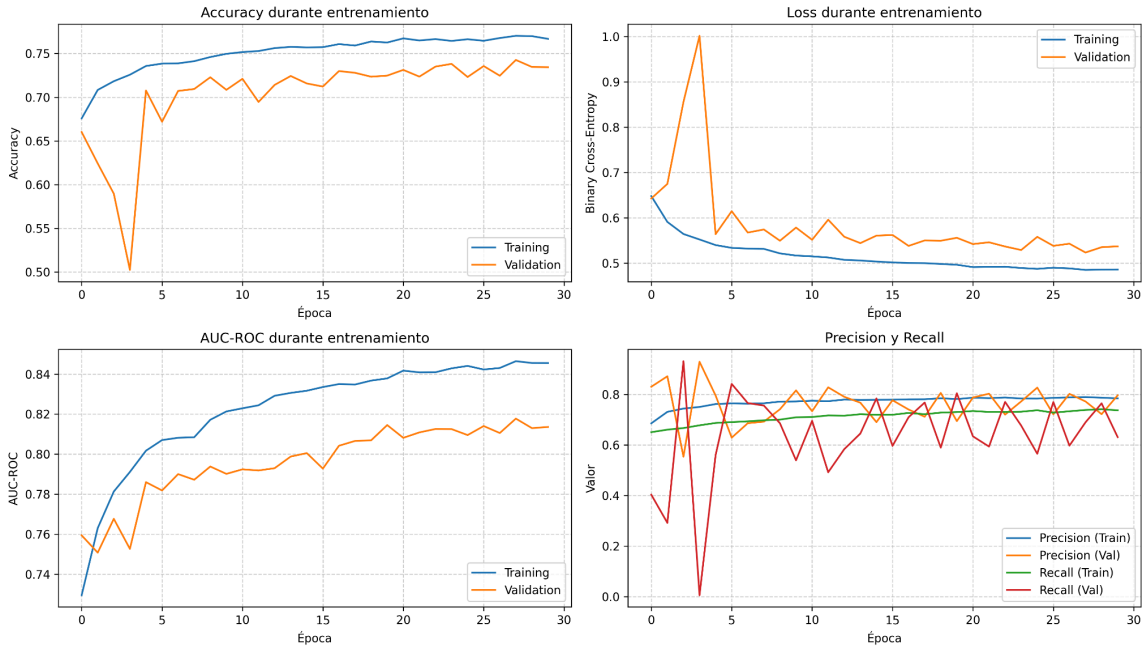
Métricas Finales

El modelo mejorado alcanzó las siguientes métricas en el conjunto de test:

Métrica	Modelo Inicial	Modelo Mejorado	Mejora Porcentual
Accuracy	67.5%	77.3%	+14.5%
AUC-ROC	0.740	0.863	+16.6%

Precision	0.693	0.843	+21.6%
Recall	0.627	0.671	+7.0%
F1-Score	0.659	0.747	+13.4%
Especificidad	No medida	0.875	N/A

Análisis de Curvas de Aprendizaje



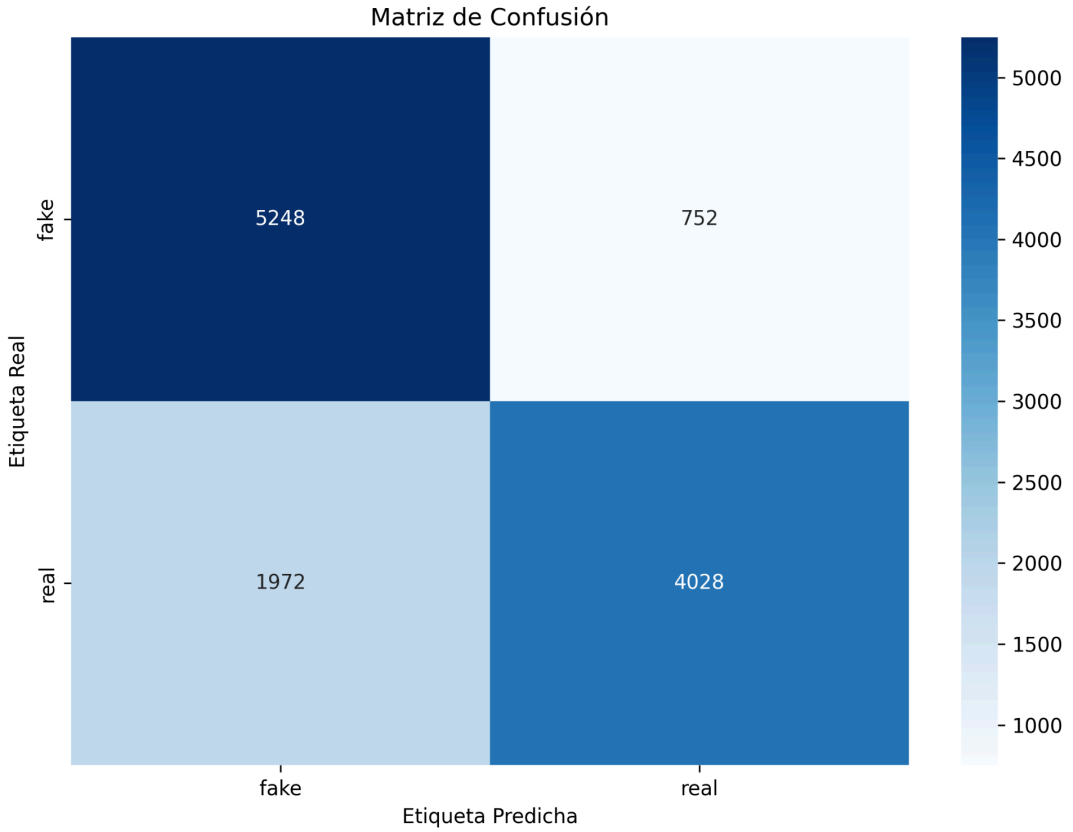
Las curvas de aprendizaje de mi modelo mejorado muestran importantes avances respecto al modelo inicial:

1. **Convergencia superior**: La accuracy de train alcanzó aproximadamente 77%, con validation estabilizándose alrededor del 73.4%, una mejora sustancial respecto al 67.5% del modelo anterior.

2. **Adaptabilidad mejorada**: El modelo se ajusta progresivamente a lo largo de las 30 épocas, con mejoras incrementales hasta las épocas finales, demostrando que funcionó la estrategia de reducción del learning rate.
3. **Métricas avanzadas**: Las gráficas adicionales de AUC-ROC y Precision/Recall muestran una mejora consistente, con el AUC-ROC de entrenamiento alcanzando 0.845 y el de validación 0.813.

Análisis de Matriz de Confusión

La matriz de confusión de mi modelo final arroja patrones interesantes en la clasificación:



Esto me permite determinar que:

1. **Asimetría en rendimiento**: El modelo es más efectivo detectando imágenes falsas (87.5% de especificidad) que reconociendo imágenes reales (67.1% de sensibilidad).
2. **Falsos negativos predominantes**: Existe un número significativamente mayor de falsos negativos (1,972) que de falsos positivos (752), indicando que mi modelo tiende a clasificar imágenes reales como falsas más frecuentemente que lo contrario.
3. **Balance por clase**: El rendimiento es más alto para la clase "fake" (F1-Score de 0.794) que para la clase "real" (F1-Score de 0.747), a pesar de que el conjunto de datos está perfectamente balanceado.

Comparación con el Estado del Arte

Aunque mi modelo mejorado (Accuracy: 77.3%, F1-Score: 0.747) no alcanza los resultados excepcionales reportados por Kroiß y Reschke (Accuracy: 96.11%, F1-Score: 0.9715), representa una mejora sustancial sobre mi modelo inicial.

Las principales diferencias que podrían explicar esta brecha incluyen:

1. **Dataset especializado**: El paper utiliza el dataset DFFD específicamente curado para esta tarea, mientras que nuestro conjunto de datos puede presentar mayor variabilidad.
2. **Ajuste de umbral**: No implementé el ajuste dinámico del umbral de clasificación (0.6587) que los autores utilizan para optimizar las métricas.
3. **Estrategia de fine-tuning**: Mi aproximación difiere del enfoque por etapas recomendado en el paper.

Además de todo, la diferencia principal entre mi modelo y el paper, es que en el paper ellos se enfocan principalmente en el reconocimiento en rostros, es decir, definir si los rostros son de humanos o generados por AI, en mi caso, fue mucho más general el análisis.

Conclusiones.

En este proyecto desarrollé un detector de imágenes falsas utilizando redes neuronales convolucionales, logrando muy buenos resultados al seguir el enfoque propuesto por Kroiß y Reschke. El modelo final, basado en una arquitectura ResNet50 con fine-tuning selectivo, alcanzó un accuracy de 77.3% y un AUC-ROC de 0.863, mejorando considerablemente frente al modelo inicial.

Las mejoras implementadas, que incluyen el descongelamiento de capas superiores, la incorporación de batch normalization, un dropout más agresivo y una estrategia adaptativa de learning rate, han demostrado ser efectivas para aumentar la capacidad del modelo para distinguir entre imágenes reales y generadas por IA.

El análisis de la matriz de confusión revela una asimetría interesante: el modelo es más efectivo detectando imágenes falsas (especificidad: 87.5%) que reconociendo imágenes reales (sensibilidad: 67.1%). Esto sugiere que las imágenes generadas artificialmente presentan patrones distintivos que el modelo identifica con mayor facilidad.

Aunque no se alcanzaron los resultados excepcionales reportados en el paper de referencia (accuracy: 96.11%), las mejoras implementadas representan un avance significativo y sientan las bases para futuros refinamientos, como la optimización del umbral de clasificación mediante análisis ROC y la exploración de arquitecturas alternativas.

Referencias.

[1] Kroiß, L., & Reschke, J. (2025). *Deepfake detection of face images based on a convolutional neural network*. arXiv. https://arxiv.org/abs/2503.11389