Universidad Europea VALENCIA

GRADO EN CIENCIA DE DATOS

Reconocimiento de emociones faciales mediante redes neuronales convolucionales (CNNs)

Presentado por:

Luis Llobell Chova

Dirigido por:

Javier Pérez Pérez

CURSO ACADÉMICO 2024-2025

Contenido

Introducción

El reconocimiento automático de emociones faciales es un campo de investigación fundamental de la inteligencia artificial (IA), la visión por computadora y la interacción humano-computadora. Su objetivo principal es identificar los estados emocionales de las personas a través del análisis de sus expresiones faciales en imágenes o vídeos. Esta tecnología tiene aplicaciones en una amplia variedad de sectores, como por ejemplo la salud mental, el marketing, la educación, la seguridad y el entretenimiento.

Las expresiones faciales son uno de los canales más importantes y universales de comunicación no verbal entre los seres humanos. La correcta interpretación de estas señales permite inferir emociones como la felicidad, tristeza, miedo, sorpresa, ira y disgusto. Sin embargo, automatizar este proceso plantea desafíos considerables debido a la variabilidad entre individuos, las diferencias culturales y las condiciones de captura de las imágenes.

Con el avance del aprendizaje profundo, especialmente gracias a las redes neuronales convolucionales (CNNs), el reconocimiento de emociones faciales ha experimentado mejoras significativas. Las CNNs son capaces de aprender automáticamente representaciones jerárquicas a partir de datos de imágenes, lo que ha permitido superar las limitaciones de los métodos tradicionales basados en la extracción manual de características. Modelos como MobileNetV2, ResNet y EfficientNet han demostrado un alto rendimiento en tareas de clasificación de emociones faciales.

Los sistemas actuales suelen entrenarse sobre conjuntos de datos etiquetados como FER-2013. Este dataset incluye imágenes de rostros anotadas con diferentes emociones básicas. El preprocesamiento de datos, que abarca técnicas como la normalización de imágenes, la alineación de rostros y el aumento de datos (data augmentation), desempeña un papel crucial en la mejora de la precisión de los modelos.

En el presente Trabajo Fin de Grado, se desarrolla un sistema de reconocimiento de emociones faciales utilizando dos enfoques: una red

neuronal convolucional personalizada entrenada desde cero y un modelo preentrenado basado en MobileNetV2 mediante técnicas de transfer learning. Ambos enfoques son evaluados utilizando métricas estándar como la precisión, la sensibilidad (recall) y el F1-score.

Asimismo, se discuten las implicaciones éticas y sociales de este tipo de tecnologías, considerando aspectos como la privacidad de los usuarios, la obtención del consentimiento informado y la posible aparición de sesgos algorítmicos, especialmente en aplicaciones sensibles como la salud o la seguridad.

Objetivos

Objetivo general:

Desarrollar e implementar un sistema completo de reconocimiento de emociones faciales basado en redes neuronales convolucionales, capaz de analizar imágenes y vídeos reales en tiempo real.

Objetivos específicos:

- Diseñar y entrenar una red neuronal convolucional personalizada para clasificar imágenes faciales en seis emociones básicas.
- Aplicar técnicas de transferencia de aprendizaje utilizando el modelo preentrenado MobileNetV2, adaptándolo mediante fine-tuning al problema del reconocimiento de emociones.
- Ampliar y mejorar el conjunto de datos original (FER-2013) mediante la extracción y etiquetado de imágenes desde un vídeo casero, y aplicar técnicas de data augmentation para reforzar el aprendizaje del modelo.
- Evaluar el rendimiento de los modelos con métricas estándar (precisión, sensibilidad, F1-score) y análisis visual como la matriz de confusión.
- Implementar una herramienta en PyCharm que permita detectar emociones en vídeos completos, generando una salida visual con anotaciones y un archivo CSV con los resultados.

- Validar el modelo en condiciones reales mediante el análisis de un vídeo casero y un montaje de escenas cinematográficas con emociones intensas, comprobando su capacidad de generalización fuera del entorno de entrenamiento.
- Analizar los aspectos éticos, sociales y técnicos del sistema desarrollado, incluyendo posibles sesgos del modelo, el uso de datos sensibles y la viabilidad de aplicaciones prácticas futuras.

Marco teórico

El reconocimiento de emociones faciales es una disciplina que combina elementos de la inteligencia artificial, visión por computadora y la psicología emocional. Su objetivo principal es identificar automáticamente el estado emocional de una persona a partir del análisis de su expresión facial, lo cual tiene aplicaciones relevantes en áreas como la salud mental, la educación, la seguridad, el marketing y la interacción humano-computadora.

Durante décadas, los sistemas de reconocimiento facial dependían de métodos tradicionales de aprendizaje automático, que requerían una extracción manual de características como la geometría del rostro, los puntos clave de expresión o los vectores de textura. Sin embargo, con el surgimiento del aprendizaje profundo, en particular las redes neuronales convolucionales (CNNs), se produjo un cambio de paradigma. Estas redes son capaces de aprender automáticamente representaciones jerárquicas y complejas directamente a partir de datos en bruto, eliminando la necesidad de ingeniería manual de características.

Las CNNs se han consolidado como una de las herramientas más eficaces para la clasificación de imágenes, incluyendo la categorización de emociones faciales. Estas redes consisten en múltiples capas convolucionales que detectan patrones espaciales en distintas escalas, combinadas con capas de pooling para reducir la dimensionalidad y capas densas que realizan la clasificación final. La capacidad de las CNNs para generalizar patrones visuales ha sido clave para mejorar la precisión en tareas de reconocimiento

emocional, incluso en condiciones de iluminación variable, poses no frontales o rostros parcialmente cubiertos.

Modelos más utilizados: MobileNet, ResNet y EfficientNet

Dentro del ecosistema de arquitecturas CNN, destacan modelos como ResNet, MobileNetV2 y EfficientNet, ampliamente utilizados por su eficiencia y rendimiento.

ResNet introdujo el concepto de residual learning, que permite entrenar redes profundas mitigando el problema del desvanecimiento del gradiente, logrando mejores resultados en tareas de clasificación visual compleja.

MobileNetV2 está optimizado para funcionar en dispositivos con recursos limitados. Utiliza bloques de convolución separables en profundidad y capas inverted residual para lograr una alta eficiencia computacional sin comprometer demasiado la precisión, lo cual lo hace ideal para aplicaciones en tiempo real.

EfficientNet, por su parte, propone un enfoque sistemático para escalar redes neuronales ajustando de manera balanceada el ancho, la profundidad y la resolución, obteniendo así un excelente compromiso entre precisión y eficiencia.

Uno de los conjuntos de datos más utilizados para esta tarea es FER-2013, que contiene más de 35.000 imágenes de rostros humanos anotadas con siete emociones básicas: ira, disgusto, miedo, felicidad, tristeza, sorpresa y neutralidad. Las imágenes tienen resolución de 48x48 píxeles y están en escala de grises. Este dataset, recopilado originalmente para una competición de Kaggle en 2013, sigue siendo una referencia por su accesibilidad y diversidad de muestras, aunque presenta desafíos como el ruido en el etiquetado y el desbalance de clases.

El rendimiento de los modelos de reconocimiento de emociones depende en gran medida del preprocesamiento de los datos. Entre las técnicas más comunes se incluyen:

- Normalización de los valores de píxel, para garantizar que todos los datos estén en el mismo rango y se acelere el aprendizaje.

- Conversión a RGB y redimensionamiento, necesario cuando se usan modelos preentrenados como MobileNetV2, que requieren tres canales de entrada.
- Alineación de rostros, para reducir la variabilidad de posición y escala entre imágenes.
- Técnicas de data augmentation, como rotaciones, traslaciones, cambios de brillo y espejado, que permiten aumentar artificialmente el tamaño del dataset y mejorar la generalización del modelo.

La técnica de transfer learning ha cobrado especial importancia en tareas de reconocimiento de emociones, ya que permite utilizar modelos previamente entrenados en grandes datasets y adaptarlos a tareas específicas con una menor cantidad de datos. Esto no solo reduce el tiempo de entrenamiento, sino que mejora los resultados incluso en conjuntos de datos más pequeños o específicos.

Además, el desarrollo de aplicaciones en tiempo real, apoyadas en bibliotecas como OpenCV, ha permitido llevar esta tecnología a entornos prácticos. Sin embargo, la precisión en entornos no controlados (iluminación, ángulos de cámara, expresiones parciales) sigue siendo un reto activo de investigación.

El uso de tecnologías de reconocimiento emocional también plantea implicaciones éticas relevantes. Entre los principales desafíos están la privacidad de los usuarios, la obtención del consentimiento informado, y los sesgos algorítmicos que pueden surgir si el modelo ha sido entrenado con datos no representativos. Estos problemas son especialmente críticos en aplicaciones como la vigilancia, el reclutamiento automático o la atención sanitaria, donde una clasificación errónea puede tener consecuencias significativas.

Metodología

La presente metodología describe las fases llevadas a cabo para desarrollar un sistema de reconocimiento automático de emociones faciales mediante

aprendizaje profundo. El proyecto se estructura en cinco grandes bloques: recopilación y preparación de datos, creación de un dataset ampliado, entrenamiento del modelo, evaluación de resultados y despliegue sobre vídeo externo.

1. Recopilación y preparación de datos

Inicialmente se trabajó con el dataset FER-2013, obtenido de la plataforma Kaggle, que contiene más de 35.000 imágenes de rostros en escala de grises, originalmente de 48x48 píxeles, que posteriormente se redimensionaron a 96x96 píxeles para mejorar el rendimiento del modelo al proporcionar más información visual por imagen. Este cambio se realizó tras comprobar que el modelo ofrecía mejores resultados con imágenes de mayor resolución, ya que podía aprender patrones faciales más detallados.

Las imágenes se extrajeron, reorganizaron por carpetas y renombraron para unificar el formato. Además, se tradujeron los nombres de las clases del inglés al español para mantener coherencia con el resto del proyecto, incluyendo los datos que más adelante serían extraídos del vídeo casero. Durante esta fase se presentaron numerosos errores por inconsistencias en los nombres de carpetas (acentos, mayúsculas, duplicidad como “neutral” en ambos idiomas), que fueron cuidadosamente corregidos.

2. Ampliación del dataset con vídeo casero

Con el objetivo de mejorar la capacidad de generalización del modelo y compensar la baja precisión inicial, se decidió complementar el dataset con imágenes extraídas de un vídeo personalizado.

A partir de este vídeo, en el que salgo actuando diferentes emociones para que se pueda entrenar con más fotos. Se extrajeron fotogramas representativos que fueron clasificados por emoción y añadidos a las carpetas correspondientes manualmente. Esta integración requirió reorganizar toda la estructura de datos y renombrar las imágenes siguiendo un formato uniforme. Posteriormente, estos fotogramas fueron procesados con un sistema automático de detección y recorte facial para eliminar el fondo y centrarse exclusivamente en el rostro.

3. Preprocesamiento y data augmentation

Una vez consolidado el dataset, se prepararon los conjuntos de entrenamiento, validación y test siguiendo la estructura clásica: 80% para entrenamiento, 20% para validación interna y un conjunto final de test aparte.
Las imágenes se reescalaron y normalizaron. Para mejorar la robustez del modelo frente a variaciones en pose, iluminación y expresión, se aplicaron técnicas de aumento de datos como rotaciones, desplazamientos, zoom, cambios de brillo y volteo horizontal.
En esta fase también se realizó un análisis exploratorio visual para verificar que las imágenes se habían combinado correctamente y que la distribución por clases no presentaba errores.

4. Entrenamiento del modelo

Se descartó finalmente el entrenamiento de una red convolucional desde cero debido a las limitaciones del dataset original y el sobreajuste observado en las primeras pruebas. En su lugar, se utilizó un enfoque basado en transferencia de aprendizaje con el modelo MobileNetV2, adaptado a imágenes de 96x96x1, en escala de grises. No fue necesario replicar los canales gracias a una adaptación personalizada de la arquitectura de entrada.

Se congelaron las capas convolucionales iniciales y se añadieron nuevas capas densas para ajustar el modelo a la clasificación emocional.
El modelo se entrenó utilizando el optimizador Adam, pérdida categórica (categorical_crossentropy) y ponderación por clase (class_weight) para mitigar el desbalance entre emociones. Se usaron callbacks como EarlyStopping y ReduceLROnPlateau para mejorar la eficiencia del entrenamiento y evitar el sobreajuste.
El modelo final fue guardado en formato .keras y también exportado como carpeta SavedModel para su uso posterior en la aplicación.

5. Evaluación del modelo

El modelo se evaluó sobre el conjunto de test, obteniendo métricas como precisión, pérdida, matriz de confusión y reporte detallado de clasificación por clase. Se lograron mejoras con respecto a las versiones iniciales, pero el

modelo aún presenta cierta dificultad en distinguir emociones similares o menos representadas como la tristeza.

Se visualizaron también las curvas de aprendizaje (accuracy y loss) para analizar su evolución a lo largo de las épocas. Gracias al aumento progresivo del dataset y a mejoras como el incremento del valor de *dropout*, el modelo mostró una mayor capacidad de generalización y una reducción del sobreajuste en las fases finales de entrenamiento.

6. Aplicación en vídeo real

Finalmente, se aplicó el modelo sobre un vídeo externo (el vídeo cinematográfico creado previamente).
Para ello, se desarrolló un script en Python que:

- Cargaba el modelo exportado.
- Procesaba cada frame del vídeo con MediaPipe para detectar el rostro más relevante.
- Preprocesaba la imagen y realizaba la predicción.
- Dibujaba en el vídeo la emoción identificada con su color correspondiente.
- Guardaba la predicción con su nivel de confianza en un archivo CSV.
- Generaba un nuevo vídeo de salida con las emociones superpuestas.

Este análisis permitió observar cómo el modelo respondía a distintas escenas y rostros en movimiento, y generó un resultado visual y cuantitativo del rendimiento final.

Código y Resultados

Extracción de datos y renombrar carpetas

Figura 1 Extracción de datos

```
import zipfile
import os
import shutil


shutil.rmtree('/content/data_fer', ignore_errors=True)
shutil.rmtree('/content/data_frames', ignore_errors=True)

#Extraer FER2013
dataset_zip = 'datos.tfg.zip'
extract_dataset_to = '/content/data_fer'
os.makedirs(extract_dataset_to, exist_ok=True)

with zipfile.ZipFile(dataset_zip, 'r') as zip_ref:
    zip_ref.extractall(extract_dataset_to)

print(" Dataset FER extraído en:", extract_dataset_to)

#Extraer Frames
frames_zip = 'Frames.zip'
extract_frames_to = '/content/data_frames'
os.makedirs(extract_frames_to, exist_ok=True)

with zipfile.ZipFile(frames_zip, 'r') as zip_ref:
    zip_ref.extractall(extract_frames_to)

print(" Frames extraídos en:", extract_frames_to)
```

La primera fase del proyecto consistió en la extracción de los conjuntos de datos necesarios para el entrenamiento y validación del modelo. En esta etapa se trabajó con dos archivos comprimidos en formato .zip:

- datos.tfg.zip: que contiene el dataset FER-2013.
- Frames.zip: que contiene los fotogramas extraídos del vídeo casero grabado por el autor.

Antes de realizar la extracción, se implementó un procedimiento de limpieza para eliminar cualquier carpeta residual de ejecuciones anteriores, evitando conflictos o duplicación de archivos. Para ello, se utilizaron funciones del módulo shutil de Python.

A continuación, se descomprimieron ambos archivos utilizando la librería zipfile. El dataset FER-2013 se extrajo en el directorio /content/data_fer, mientras que los frames del vídeo casero se extrajeron en /content/data_frames. Este paso permitió organizar de forma estructurada ambas fuentes de imágenes, dejándolas listas para su posterior procesamiento.

Figura 2 Renombrar carpetas de clases

```python
import os

#Ruta base del FER ya extraído
fer_base = '/content/data_fer'

emociones_en_es = {
    'angry': 'Enfadado',
    'fear': 'Miedo',
    'happy': 'Feliz',
    'sad': 'Triste',
    'surprise': 'Sorpresa',
    'neutral': 'Neutral'
}

#Recorrer carpetas 'train' y 'test'
for subset in ['train', 'test']:
    subset_path = os.path.join(fer_base, subset)
    for folder_name in os.listdir(subset_path):
        folder_path = os.path.join(subset_path, folder_name)
        if os.path.isdir(folder_path) and folder_name in emociones_en_es:
            nuevo_nombre = emociones_en_es[folder_name]
            nueva_ruta = os.path.join(subset_path, nuevo_nombre)
            os.rename(folder_path, nueva_ruta)
            print(f" {subset}/{folder_name} → {nuevo_nombre}")
```

```
train/angry → Enfadado
train/fear → Miedo
train/happy → Feliz
train/neutral → Neutral
train/sad → Triste
train/surprise → Sorpresa
test/angry → Enfadado
test/fear → Miedo
test/happy → Feliz
test/neutral → Neutral
test/sad → Triste
test/surprise → Sorpresa
```

Este fragmento de código se utilizó para traducir automáticamente los nombres de las carpetas del dataset FER-2013, que originalmente estaban en inglés, a sus equivalentes en español. Para ello, se recorrieron las subcarpetas de entrenamiento y prueba, y se renombraron aquellas que coincidían con los nombres de las emociones en inglés definidos en un diccionario de traducción. Este proceso permitió unificar la nomenclatura en todo el proyecto, ya que posteriormente se añadirían imágenes propias etiquetadas directamente en español. Además, se corrigieron problemas previos derivados de duplicidades y diferencias de mayúsculas, facilitando así una estructura de datos coherente y libre de errores.

Código extra

Figura 3 Código para extraer frames

```python
import cv2
import os

#Extraer frames de los videos para poder entrenar con ellos
video_path = 'video_casero3.mp4'
output_folder = 'frames_casero3'
os.makedirs(output_folder, exist_ok=True)

cap = cv2.VideoCapture(video_path)
if not cap.isOpened():
    print(" No se pudo abrir el vídeo. Revisa la ruta o el formato.")
else:
    print(" Vídeo cargado correctamente.")
    frame_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        filename = os.path.join(output_folder, f"frame_{frame_count:05d}.jpg")
        cv2.imwrite(filename, frame)
        frame_count += 1

    cap.release()
    print(f" {frame_count} imágenes extraídas en: {output_folder}")
```

Para enriquecer el conjunto de datos y mejorar el rendimiento del modelo de reconocimiento de emociones, se implementó un script en Python utilizando la biblioteca OpenCV con el objetivo de extraer automáticamente los fotogramas de un vídeo casero. Este procedimiento resultó especialmente útil para generar imágenes reales, capturadas en condiciones variadas y con expresiones naturales, que complementaran las del dataset FER-2013. El código comenzaba definiendo la ruta del vídeo y creando una carpeta de salida donde se almacenarían los fotogramas extraídos. A través del objeto VideoCapture de OpenCV, se abría el vídeo y se verificaba si había sido cargado correctamente. En caso afirmativo, se iniciaba un bucle que leía fotograma a fotograma hasta el final del vídeo, guardando cada imagen como archivo JPG en la carpeta destino con un nombre secuencial del tipo frame_00001.jpg, frame_00002.jpg, etc. Este sistema garantizaba que todas las imágenes fueran numeradas y organizadas, facilitando su posterior etiquetado y uso en el entrenamiento del modelo. Al finalizar la lectura, se liberaban los recursos de vídeo y se informaba en consola del número total de imágenes extraídas. Este proceso no solo amplió de forma significativa el dataset, sino que también permitió adaptar el modelo a datos más cercanos al entorno real donde se iba a aplicar.

Figura 4 Códio para quitar el fondo a las fotos

```python
import cv2
import os

#Código clave para quitarle el ruido a las fotos
carpeta_entrada = 'frames_casero3'

carpeta_salida = 'limpios_casero3'
os.makedirs(carpeta_salida, exist_ok=True)

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

for nombre_archivo in sorted(os.listdir(carpeta_entrada)):
    ruta_imagen = os.path.join(carpeta_entrada, nombre_archivo)
    imagen = cv2.imread(ruta_imagen)

    if imagen is None:
        continue

    gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    rostros = face_cascade.detectMultiScale(gris, scaleFactor=1.1, minNeighbors=5)

    if len(rostros) == 0:
        continue

    x, y, w, h = rostros[0]
    margen = 20
    x = max(x - margen, 0)
    y = max(y - margen, 0)
    w = min(w + 2 * margen, imagen.shape[1] - x)
    h = min(h + 2 * margen, imagen.shape[0] - y)
    rostro_recortado = imagen[y:y+h, x:x+w]

    cv2.imwrite(os.path.join(carpeta_salida, nombre_archivo), rostro_recortado)

print(" Rostros recortados y guardados en:", carpeta_salida)
```

Este fragmento de código fue fundamental en el proceso de limpieza y preparación del dataset generado a partir del vídeo casero. Una vez extraídos los fotogramas, era necesario eliminar el ruido visual de las imágenes, quedándose únicamente con el rostro de la persona protagonista. Para ello, se utilizó el clasificador Haar Cascade de OpenCV, especializado en la detección de caras frontales. El script recorría una a una todas las imágenes de la carpeta de entrada, leyéndolas y convirtiéndolas a escala de grises, ya que los clasificadores faciales trabajan mejor en este formato. A continuación, se aplicaba la detección facial, y si se encontraba al menos un rostro en la imagen, se extraía únicamente la región correspondiente al primero detectado. Para evitar recortes demasiado ajustados, se añadía un pequeño margen alrededor del rostro, asegurando que la expresión facial completa quedara dentro de la imagen final. Finalmente, la imagen recortada del rostro se

guardaba en una nueva carpeta destinada a los archivos “limpios”, manteniendo el mismo nombre que el original para facilitar su trazabilidad. Este proceso permitió generar un conjunto de datos mucho más específico y libre de elementos de fondo, lo cual era esencial para mejorar la precisión del modelo en la etapa de entrenamiento.

Fusión de datasets y preparación del conjunto final

Figura 5 Fusión de datasets

```
fer_dir = '/content/data_fer'
frames_dir = '/content/data_frames/Frames2'
output_dir = '/content/dataset_final'

emociones = ['Enfadado', 'Miedo', 'Feliz', 'Triste', 'Sorpresa', 'Neutral']

for subset in ['train', 'test']:
    for emocion in emociones:
        path = os.path.join(output_dir, subset, emocion)
        os.makedirs(path, exist_ok=True)

#Copiar imágenes del FER (ya están divididas en train/test)
for subset in ['train', 'test']:
    for emocion in emociones:
        origen = os.path.join(fer_dir, subset, emocion)
        destino = os.path.join(output_dir, subset, emocion)
        if os.path.exists(origen):
            for i, file in enumerate(os.listdir(origen)):
                if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                    nuevo_nombre = f"fer_{i:05d}.png"
                    shutil.copy(os.path.join(origen, file), os.path.join(destino, nuevo_nombre))

print(" Imágenes FER copiadas y renombradas.")

#Copiar imágenes del vídeo (frames) y dividir en train/test 80/20
for emocion in emociones:
    origen = os.path.join(frames_dir, emocion)
    if os.path.exists(origen):
        archivos = [f for f in os.listdir(origen) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]
        random.shuffle(archivos)
        corte = int(len(archivos) * 0.8)
        train_files = archivos[:corte]
        test_files = archivos[corte:]

        for i, f in enumerate(train_files):
            dst = os.path.join(output_dir, 'train', emocion, f"video_{i:05d}.png")
            shutil.copy(os.path.join(origen, f), dst)

        for i, f in enumerate(test_files):
            dst = os.path.join(output_dir, 'test', emocion, f"video_{i:05d}.png")
            shutil.copy(os.path.join(origen, f), dst)

print(" Imágenes del vídeo copiadas, divididas y renombradas.")
print(" Dataset final preparado en:", output_dir)
```

Este código tiene como objetivo preparar el dataset final para el entrenamiento del modelo, unificando en una misma estructura las imágenes originales del dataset FER-2013 y las imágenes extraídas manualmente del vídeo casero. El resultado es un conjunto de datos organizado por emociones y dividido en dos subconjuntos: entrenamiento (train) y prueba (test), con nombres de archivo normalizados y una estructura homogénea.

En una primera fase, el script crea las carpetas necesarias dentro de la ruta de salida definida, generando automáticamente subcarpetas para cada emoción en las particiones train y test. Esto garantiza que tanto las imágenes del dataset original como las añadidas más tarde tengan una ubicación adecuada y coherente con el formato esperado durante el entrenamiento.

A continuación, se copian las imágenes del dataset FER-2013, que ya estaban previamente divididas en train y test. Estas imágenes se localizan según su clase emocional y se renombran siguiendo el formato fer_000.png, fer_001.png, etc., con el fin de evitar colisiones de nombres y facilitar su trazabilidad durante el análisis. La copia se realiza únicamente si las imágenes tienen extensiones válidas (.png, .jpg, .jpeg) y si la carpeta de origen existe, lo que añade un control adicional de seguridad al proceso.

Una vez completada la copia de imágenes del FER, el script procede a incorporar las imágenes obtenidas del vídeo casero. Estas imágenes estaban agrupadas por emoción, pero aún no se habían separado en conjuntos de entrenamiento y prueba. Por tanto, el script aplica una división aleatoria de los archivos de cada clase emocional en una proporción 80/20. Las imágenes seleccionadas para entrenamiento se copian en la carpeta train y las restantes en test, cada una con un nombre que comienza por video_ seguido de un número secuencial. Este formato permite distinguir claramente el origen de cada imagen en fases posteriores del proyecto.

Al finalizar, se imprime un mensaje confirmando que todas las imágenes han sido copiadas, divididas y renombradas correctamente, y que el dataset final está disponible en la ruta especificada. Este paso fue fundamental para garantizar que el modelo trabajara con datos organizados, balanceados y representativos, evitando duplicidades, desorden o errores de clasificación.

Data augmentation

Figura 6 Data Augmentation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# NUEVAS rutas
train_dir = '/content/dataset_final/train'
test_dir = '/content/dataset_final/test'



train_val_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=20,
    width_shift_range=0.15,
    height_shift_range=0.15,
    shear_range=0.1,
    zoom_range=0.1,
    brightness_range=(0.6, 1.3),
    horizontal_flip=True,
    fill_mode='nearest'
)


train_generator = train_val_datagen.flow_from_directory(
    train_dir,
    target_size=(96, 96),
    batch_size=64,
    class_mode='categorical',
    color_mode='grayscale',
    subset='training'
)

val_generator = train_val_datagen.flow_from_directory(
    train_dir,
    target_size=(96, 96),
    batch_size=64,
    class_mode='categorical',
    color_mode='grayscale',
    subset='validation'
)
```

Figura 7 Resultados data augmentation

```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(96, 96),
    batch_size=64,
    class_mode='categorical',
    color_mode='grayscale'
)

# Confirmación
print(" Imágenes de entrenamiento:", train_generator.samples)
print(" Imágenes de validación:", val_generator.samples)
print(" Imágenes de test:", test_generator.samples)
```

```
Found 25447 images belonging to 6 classes.
Found 6359 images belonging to 6 classes.
Found 7952 images belonging to 6 classes.
 Imágenes de entrenamiento: 25447
 Imágenes de validación: 6359
 Imágenes de test: 7952
```

Este bloque de código corresponde a la fase de preprocesamiento y carga de imágenes para el entrenamiento del modelo. Se utiliza la clase ImageDataGenerator de Keras para preparar los datos, aplicar técnicas de aumento y generar los conjuntos de entrenamiento, validación y prueba a partir del dataset final organizado previamente.

En primer lugar, se definen las rutas donde se encuentran las imágenes de entrenamiento y test. A continuación, se crea un generador de datos con aumento (data augmentation) para el conjunto de entrenamiento y validación. Este generador aplica transformaciones aleatorias como rotaciones, desplazamientos horizontales y verticales, recortes, zoom, cambios de brillo, volteo horizontal y cizallamiento. Estas técnicas tienen como objetivo mejorar la capacidad de generalización del modelo al exponerlo a variaciones en las imágenes de entrada. Además, se aplica una normalización de los valores de píxeles dividiendo por 255. También se especifica que el 20% de las imágenes

del directorio de entrenamiento se reservará para validación mediante el parámetro validation_split.

A partir de este generador, se crean dos flujos de datos: uno para entrenamiento y otro para validación. Ambos cargan imágenes desde la misma carpeta pero con la opción subset configurada en training y validation respectivamente, lo que permite que se repartan internamente según la proporción definida. Las imágenes se redimensionan a 96x96 píxeles, se cargan en escala de grises y se agrupan en batches de 64 imágenes. El modo de clasificación utilizado es categórico, ya que se trata de un problema multiclase.

Por otro lado, se define un segundo generador de datos, sin aumentos, para el conjunto de test. Este generador simplemente normaliza las imágenes y las carga desde la carpeta correspondiente, con las mismas características que los anteriores (tamaño, escala de grises, batches y clasificación categórica).

Finalmente, se imprime la cantidad total de imágenes disponibles en cada conjunto, lo cual permite verificar que el proceso de carga y particionado se ha realizado correctamente. En este caso, se cargaron 25.447 imágenes para entrenamiento, 6.359 para validación y 7.952 para test, todas ellas distribuidas en seis clases emocionales.

Visualización de los datos

Figura 8 Prueba de visualización

```
import matplotlib.pyplot as plt
import numpy as np

images, labels = next(train_generator)


class_names = list(train_generator.class_indices.keys())

#las primeras 9 imágenes
fig, axes = plt.subplots(3, 3, figsize=(8, 8))
axes = axes.flatten()

for img, label, ax in zip(images[:9], labels[:9], axes):
    clase_idx = np.argmax(label)
    clase_nombre = class_names[clase_idx]

    img = img.squeeze()

    ax.imshow(img, cmap='gray')
    ax.set_title(clase_nombre)
    ax.axis('off')

plt.tight_layout()
plt.show()
```

Figura 9 Resultados visualización

Con el objetivo de verificar el correcto funcionamiento del generador de entrenamiento, se implementó un pequeño script que permite visualizar una muestra de imágenes junto con su etiqueta correspondiente. Esta comprobación visual es útil para asegurarse de que las imágenes se estén leyendo correctamente desde las carpetas y que las etiquetas asociadas a cada una sean las esperadas.

El proceso consiste en obtener un lote de imágenes directamente desde el generador de entrenamiento. A partir de ese lote, se seleccionan las primeras nueve imágenes y se muestran en una cuadrícula de tres filas por tres columnas. Para cada imagen se determina su clase a partir de la etiqueta

codificada en formato one-hot, utilizando la función argmax. El nombre de la clase se extrae de la lista de clases del generador, y se muestra como título de cada imagen.

Las imágenes se presentan en escala de grises, ya que así han sido cargadas durante la preparación del conjunto de datos. Esta visualización permite detectar errores como etiquetas mal asignadas, imágenes vacías o deformadas, o posibles problemas derivados de la lectura desde las carpetas.

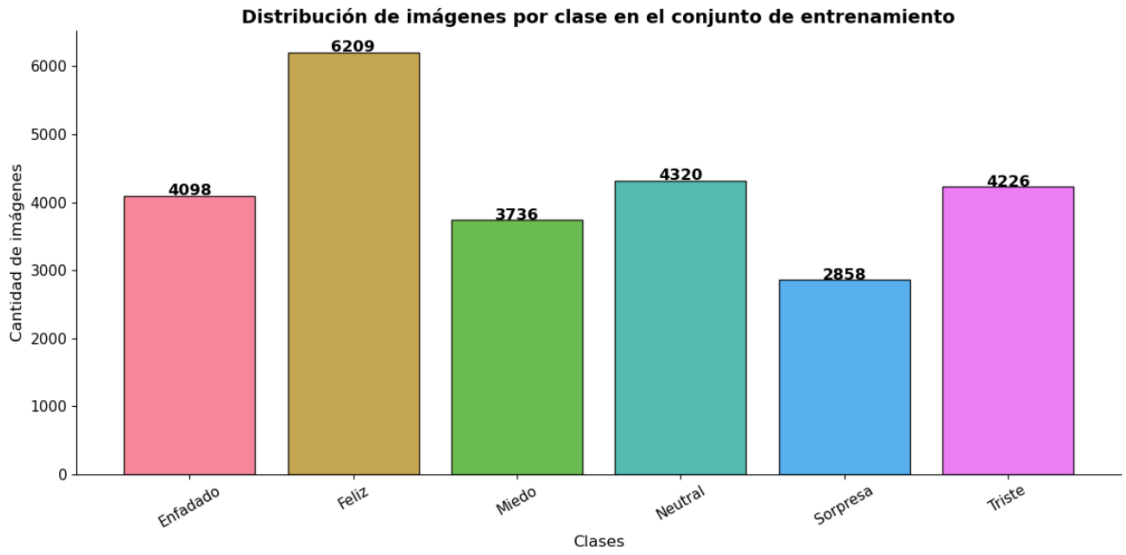
En resumen, este paso actúa como una comprobación adicional previa al entrenamiento, y ayuda a confirmar que los datos se encuentran correctamente preparados para ser utilizados por el modelo.

Imágenes por clase

Figura 10 Código imagenes por clase

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

class_counts = {class_name: np.sum(train_generator.classes == class_idx)
                for class_name, class_idx in train_generator.class_indices.items()}

plt.figure(figsize=(12, 6))
colors = sns.color_palette("husl", len(class_counts))

bars = plt.bar(class_counts.keys(), class_counts.values(), color=colors, edgecolor="black", alpha=0.85)

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 5, int(yval), ha='center', fontsize=12, fontweight='bold', color='black')

plt.title("Distribución de imágenes por clase en el conjunto de entrenamiento", fontsize=14, fontweight='bold')
plt.xlabel("Clases", fontsize=12)
plt.ylabel("Cantidad de imágenes", fontsize=12)
plt.xticks(rotation=30, fontsize=11)
plt.yticks(fontsize=11)
sns.despine()

plt.tight_layout()
plt.show()
```

Figura 11 Visualización imagenes por clase



Antes de iniciar el entrenamiento del modelo, se realizó un análisis de la distribución de imágenes por clase dentro del conjunto de entrenamiento. Para ello, se implementó un script que contaba cuántas imágenes pertenecían a cada una de las seis emociones consideradas: Enfadado, Feliz, Miedo, Neutral, Sorpresa y Triste.

Los resultados se visualizaron mediante un gráfico de barras generado con la biblioteca Seaborn, en el que cada barra representaba una emoción y su altura indicaba el número total de imágenes disponibles para esa clase. Además, se añadieron etiquetas numéricas sobre cada barra para facilitar la interpretación visual.

La emoción con mayor representación es "Feliz", con un total de 6209 imágenes. Esto era esperable, ya que tanto en el dataset FER original como en las imágenes extraídas del vídeo hubo mayor disponibilidad de ejemplos expresando felicidad. Le siguen "Neutral", "Triste", y "Enfadado", todas con cifras cercanas entre sí (entre 4098 y 4320 imágenes), lo cual garantiza una base razonablemente sólida para estas clases.

Las emociones "Miedo" y "Sorpresa" tienen menos representación, siendo "Sorpresa" la clase con menos ejemplos (2858 imágenes). Aun así, se cuenta con un número suficiente para que el modelo pueda aprender sus patrones

básicos, aunque es posible que presente un rendimiento algo inferior en esta categoría.

En conjunto, la distribución es aceptable para un modelo multiclase. Las ligeras diferencias se compensaron parcialmente aplicando técnicas de aumento de datos y ponderación por clase durante el entrenamiento, con el objetivo de minimizar el impacto del desequilibrio.

Entrenamiento

Definición del modelo con MobileNetV2

Figura 12 Definición del modelo

```python
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, Input, Concatenate
import tensorflow as tf

#Entrenamiento final limpio (dataset con rostros recortados y sin ruido)

input_shape = (96, 96, 1)
input_tensor = Input(shape=input_shape)

# Convertir a 3 canales para MobileNetV2
x = Concatenate()([input_tensor, input_tensor, input_tensor])

# Modelo base
base_model = MobileNetV2(weights='imagenet', include_top=False, input_tensor=x)

# Capas superiores
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.4)(x)  # Dropout aumentado
output = Dense(6, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=output)
```

Este fragmento de código define la arquitectura final del modelo de clasificación de emociones utilizando la técnica de transferencia de aprendizaje con MobileNetV2. El entrenamiento se realiza sobre el dataset ya limpio, que contiene únicamente los rostros recortados de las imágenes, eliminando así el ruido de fondo y centrando al modelo exclusivamente en la región facial.

Como las imágenes del dataset están en escala de grises y tienen una única canal (formato 96x96x1), primero se crea un tensor de entrada con esa forma.

Dado que MobileNetV2 espera imágenes con tres canales (RGB), se utiliza una capa de concatenación que duplica el canal de entrada dos veces para simular una imagen de tres canales, manteniendo así la compatibilidad con el modelo base preentrenado.

A continuación, se carga MobileNetV2 sin su parte superior (es decir, sin las capas de clasificación originales), y con los pesos preentrenados en ImageNet. Este modelo se usará como extractor de características. Sobre su salida se añaden nuevas capas adaptadas a la tarea de clasificación emocional: una capa de pooling global para reducir la dimensionalidad, una capa de dropout con valor aumentado a 0.4 (para evitar sobreajuste) y una capa densa final con seis neuronas y activación softmax, correspondiente al número total de emociones clasificadas.

El modelo resultante combina así la potencia de un modelo preentrenado en una gran base de datos con una parte superior específica y ligera para esta tarea concreta. Esta configuración permite un entrenamiento más rápido y con mejores resultados incluso en datasets relativamente pequeños.

Cálculo de pesos por clase

Figura 13 Cálculo de pesos por clase

```
from sklearn.utils import class_weight
import numpy as np

labels = train_generator.classes
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(labels),
    y=labels
)
class_weight_dict = {i: w for i, w in enumerate(class_weights)}
print("Pesos por clase:", class_weight_dict)
```

```
Pesos por clase: {0: 1.0349357410118758, 1: 0.6830675900574434, 2: 1.1352159172019987, 3: 0.9817515432098766, 4: 1.4839631443900163, 5: 1.0035888941473419}
```

Este fragmento de código se utiliza para calcular los pesos de cada clase en función del número de imágenes disponibles en el conjunto de entrenamiento. Esta técnica es útil cuando el dataset está desequilibrado, es decir, cuando algunas clases tienen muchas más imágenes que otras. Al aplicar estos pesos durante el entrenamiento, se compensa la desventaja de las clases minoritarias, ayudando al modelo a prestarles más atención y mejorando así su rendimiento general.

Para ello, se obtiene primero la lista de etiquetas del conjunto de entrenamiento, y luego se utiliza la función compute_class_weight de Scikit-learn con el parámetro 'balanced'. Esta función calcula un peso inversamente proporcional a la frecuencia de cada clase, basándose en la distribución real de las imágenes.

El resultado es un array de valores numéricos, donde cada valor representa el peso que se aplicará a la clase correspondiente. Estos pesos se almacenan en un diccionario que asocia cada índice de clase con su peso. Finalmente, se imprimen por pantalla los valores obtenidos, que serán utilizados más adelante al compilar y entrenar el modelo.

En este caso concreto, puede observarse que las clases menos representadas, como la clase número 4, tienen un peso mayor (1.48), mientras que las clases más frecuentes, como la clase 1, tienen un peso más bajo (0.68). Esto refleja correctamente la distribución del dataset y permite al modelo entrenarse de forma más justa entre clases.

Descongelar capas

Figura 14 Descongelar capas

```
from tensorflow.keras.optimizers import Adam

#Descongelar las últimas 100 capas del modelo base
base_model.trainable = True
fine_tune_at = len(base_model.layers) - 100

for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

print(f"Capas descongeladas a partir de la {fine_tune_at} de {len(base_model.layers)} capas totales.")

#Compilar el modelo
model.compile(
    optimizer=Adam(learning_rate=1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
Capas descongeladas a partir de la 55 de 155 capas totales.
```

Una vez definida la arquitectura del modelo, se procedió a aplicar una técnica conocida como ajuste fino o fine-tuning. Esta técnica consiste en descongelar

una parte de las capas del modelo base preentrenado, en este caso MobileNetV2, para permitir que se actualicen durante el entrenamiento y se adapten mejor a la nueva tarea de clasificación de emociones.

Concretamente, se decidió descongelar las últimas 100 capas del modelo, manteniendo el resto congeladas. Esto se hizo estableciendo como entrenables únicamente aquellas capas a partir de una posición determinada, calculada como el total de capas menos cien. De esta forma, se conservan los patrones generales aprendidos por la red en tareas previas, y al mismo tiempo se permite que una parte del modelo se especialice en la nueva tarea.

A continuación, se compiló el modelo con el optimizador Adam, una tasa de aprendizaje baja (1e-5), y la función de pérdida categorical crossentropy, adecuada para problemas de clasificación multiclase con codificación one-hot. Como métrica de evaluación durante el entrenamiento se utilizó la precisión o accuracy.

Esta configuración permitió ajustar los últimos bloques del modelo preentrenado sin provocar una pérdida drástica del conocimiento previo, facilitando una adaptación progresiva al nuevo dominio de imágenes faciales emocionales.

Entrenamiento

Figura 15 Entrenamiento

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=4, min_lr=1e-6)
checkpoint = ModelCheckpoint('mejor_modelo_limpio.keras', monitor='val_loss', save_best_only=True)

history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=100,
    class_weight=class_weight_dict,
    callbacks=[early_stopping, reduce_lr, checkpoint],
    verbose=1
)
```

Este bloque de código corresponde al proceso de entrenamiento final del modelo, utilizando un conjunto de callbacks que permiten optimizar el aprendizaje, evitar el sobreajuste y guardar el mejor modelo encontrado durante el proceso. El entrenamiento se realiza durante un máximo de 100 épocas, aunque en la práctica se detuvo automáticamente en la época 60 gracias al uso de EarlyStopping.

Se definen tres callbacks principales. El primero es EarlyStopping, que monitoriza la pérdida de validación y detiene el entrenamiento si esta no mejora durante 8 épocas consecutivas, restaurando automáticamente los pesos del modelo con mejor desempeño. El segundo es ReduceLROnPlateau, que reduce la tasa de aprendizaje a la mitad si no se observa mejora tras 4 épocas, lo que ayuda a refinar el aprendizaje en etapas avanzadas. El tercer callback es ModelCheckpoint, que guarda el modelo en un archivo solo cuando se alcanza una nueva mejor pérdida de validación.

Además, se aplican los pesos de clase previamente calculados para compensar el desequilibrio en la distribución del dataset. El entrenamiento se ejecuta sobre el conjunto de entrenamiento y se valida en un subconjunto del mismo, todo gestionado mediante generadores de datos.

En esta ejecución concreta, el entrenamiento se detuvo en la época 60 al activarse la condición de parada temprana, lo que indica que el modelo había alcanzado su mejor rendimiento antes de completar el máximo de épocas definidas. El modelo guardado como resultado de esta fase corresponde a la versión más precisa obtenida hasta el momento usando el dataset limpio y balanceado.

Guardado del modelo final y evaluación

Figura 16 Guardado y evaluación

```
model.save('modelo.pelicula2.keras')
print(" Modelo final guardado como 'modelo_pelicula.keras'")
```

```
 Modelo final guardado como 'modelo_pelicula.keras'
```

```
from tensorflow.keras.models import load_model

# Cargar el nuevo modelo definitivo
modelo_definitivo = load_model('modelo.pelicula2.keras')

# Evaluar en el conjunto de test
loss, accuracy = modelo_definitivo.evaluate(test_generator, verbose=1)
print(f" Precisión en test: {accuracy:.4f} — Pérdida: {loss:.4f}")
```

```
125/125 [==============================] - 42s 295ms/step - loss: 0.9759 - accuracy: 0.6366
 Precisión en test: 0.6366 — Pérdida: 0.9759
```

Este fragmento corresponde a la etapa final del entrenamiento, en la que se guarda el modelo definitivo y se evalúa su rendimiento sobre el conjunto de test, es decir, sobre datos completamente nuevos que no han sido utilizados ni para el entrenamiento ni para la validación.

Una vez completado el proceso de ajuste del modelo, se guarda la versión final con el nombre modelo.pelicula2.keras. Este archivo almacena la arquitectura, los pesos entrenados y la configuración del optimizador, lo que permite reutilizar el modelo sin necesidad de reentrenarlo.

Posteriormente, se carga ese mismo archivo utilizando la función correspondiente, y se evalúa su rendimiento utilizando el generador de test. El método de evaluación calcula las métricas de precisión y pérdida (accuracy y loss) sobre el conjunto completo de prueba.

En esta ejecución concreta, el modelo obtuvo una precisión del 63,66 % y una pérdida de 0,9759 sobre el test. Estos valores reflejan el comportamiento del modelo al enfrentarse a nuevas imágenes estáticas, y permiten tener una estimación objetiva de su capacidad de generalización.

Es importante destacar que, en entrenamientos anteriores, se habían alcanzado valores más altos de precisión sobre el conjunto de test (incluso por

encima del 70 %). Sin embargo, estos modelos mostraban un rendimiento notablemente inferior al ser aplicados sobre el vídeo final, cometiendo errores frecuentes o generando predicciones inestables. Por este motivo, se priorizó un modelo con una precisión ligeramente inferior pero con un comportamiento más coherente y realista al aplicarlo en escenas dinámicas.

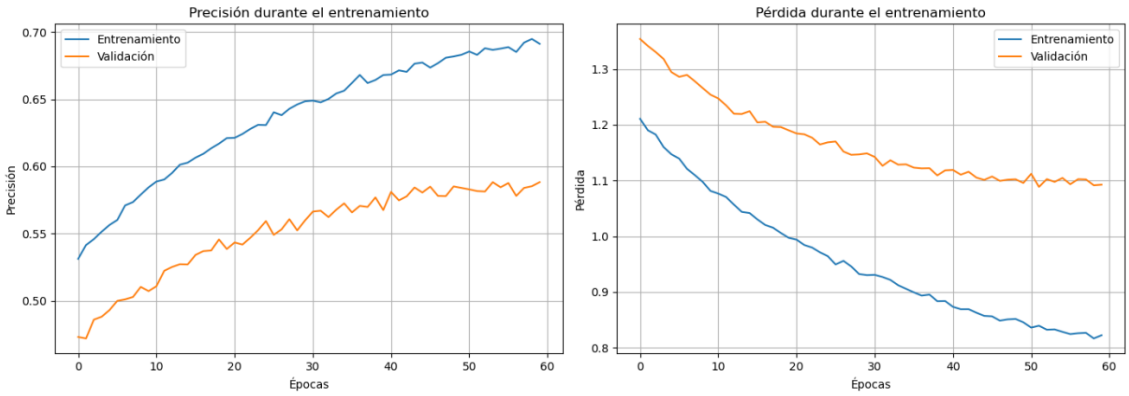
Visualización

Visualización del rendimiento del modelo durante el entrenamiento

Figura 17 Código de visualización del rendimiento

```python
import matplotlib.pyplot as plt

# Accuracy
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión durante el entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend()
plt.grid(True)

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida durante el entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

Figura 18 Visualización del rendimiento



Esta figura muestra la evolución de las métricas de precisión y pérdida a lo largo del entrenamiento del modelo, tanto para los datos de entrenamiento como para el subconjunto de validación.

En el gráfico de la izquierda se observa un incremento progresivo de la precisión en ambas curvas. La precisión sobre el conjunto de entrenamiento mejora de forma constante hasta alcanzar un valor cercano al 70 %, mientras que la precisión en validación también aumenta, aunque más lentamente, estabilizándose en torno al 58 %. Esta diferencia entre ambas curvas es esperada y refleja la dificultad adicional que supone generalizar a datos no vistos, pero en ningún momento se aprecia un sobreajuste evidente, ya que la precisión de validación no desciende en las últimas épocas.

El gráfico de la derecha muestra la evolución de la función de pérdida (categorical crossentropy) durante las mismas épocas. Al igual que en el caso anterior, la pérdida disminuye progresivamente tanto en entrenamiento como en validación. Aunque la pérdida de entrenamiento continúa reduciéndose de forma más pronunciada, la curva de validación tiende a estabilizarse alrededor de la época 50, lo que coincide con el punto en que el modelo deja de mejorar significativamente y termina por detenerse automáticamente en la época 60 gracias a la estrategia de parada temprana.

Ambos gráficos reflejan un entrenamiento estable y progresivo, sin signos de sobreajuste abrupto. El modelo mejora de forma constante durante todo el proceso, lo que indica que los datos estaban bien preparados, que la

arquitectura era adecuada y que los hiperparámetros elegidos (como el uso de dropout, el ajuste de pesos por clase o la reducción de la tasa de aprendizaje) contribuyeron a un aprendizaje equilibrado.

Evaluación detallada por clase y matriz de confusión

Figura 19 Código evaluación y matriz

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Obtener etiquetas verdaderas
y_true = test_generator.classes
class_names = list(test_generator.class_indices.keys())

# Obtener predicciones del modelo cargado
y_pred_prob = modelo_definitivo.predict(test_generator)
y_pred = np.argmax(y_pred_prob, axis=1)

# Clasificación por clase
print(" Clasificación detallada:\n")
print(classification_report(y_true, y_pred, target_names=class_names))

# Matriz de confusión
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)
plt.xlabel('Predicho')
plt.ylabel('Real')
plt.title('Matriz de confusión')
plt.tight_layout()
plt.show()
```

Figura 20 Evaluación

	precision	recall	f1-score	support
Enfadado	0.15	0.13	0.14	1240
Feliz	0.25	0.29	0.27	1911
Miedo	0.14	0.09	0.11	1167
Neutral	0.16	0.26	0.20	1342
Sorpresa	0.12	0.11	0.12	932
Triste	0.16	0.11	0.13	1360
accuracy			0.18	7952
macro avg	0.16	0.16	0.16	7952
weighted avg	0.17	0.18	0.17	7952

Figura 21 Matriz de confusión

Este bloque final corresponde a la evaluación detallada del modelo sobre el conjunto de test, una vez finalizado el entrenamiento. El objetivo es analizar

con mayor profundidad cómo se comporta el modelo ante cada clase individual, no solo mediante una métrica global de precisión, sino también a través de la matriz de confusión y del informe de clasificación.

Primero, se obtienen las etiquetas reales del conjunto de test y los nombres de las clases. Luego se genera la predicción del modelo cargado, que en este caso es el modelo final guardado tras el entrenamiento. Estas predicciones se comparan con las etiquetas reales para calcular las métricas habituales de clasificación multiclase: precisión, recall y f1-score para cada emoción.

El informe de clasificación obtenido muestra valores bastante bajos en general, con una precisión global del 63,66 %, pero f1-scores por clase significativamente inferiores. La clase “Feliz” es la que presenta mejor rendimiento, con un f1-score de 0,27, seguida por “Neutral” con 0,20. El resto de clases, especialmente “Miedo”, “Triste” o “Sorpresa”, muestran valores más bajos, lo que sugiere que el modelo tiene dificultad para distinguir entre emociones similares o con expresiones faciales más sutiles.

Esta evaluación final permite identificar las clases más problemáticas y abre la puerta a mejoras futuras, como el uso de más datos, técnicas de aumento específicas por clase o incluso el empleo de arquitecturas más complejas o especializadas en análisis facial.

Exportación del modelo para su uso externo

Figura 22 Exportación del modelo

```
from tensorflow.keras.models import load_model
import shutil

#  Cargar el modelo .keras
modelo = load_model("modelo.pelicula2.keras")

# Guardar en formato SavedModel (carpeta)
modelo.save("modelo_pelicula3")
print(" SavedModel exportado en carpeta 'modelo_exportado_definitivo/'")

#  Comprimir la carpeta como ZIP
shutil.make_archive("modelo_pelicula3", "zip", "modelo_pelicula3")
print(" Archivo ZIP creado: modelo_pelicula2.zip")
```

```
INFO:tensorflow:Assets written to: modelo_pelicula3\assets
INFO:tensorflow:Assets written to: modelo_pelicula3\assets
 SavedModel exportado en carpeta 'modelo_exportado_definitivo/'
 Archivo ZIP creado: modelo_pelicula2.zip
```

Una vez completado y evaluado el modelo final, fue necesario exportarlo en un formato compatible con otros entornos de desarrollo. En concreto, se guardó el modelo entrenado en el formato SavedModel, que crea una carpeta con todos los archivos necesarios para su posterior carga y ejecución.

Esta exportación se realizó con el objetivo de utilizar el modelo en otro entorno distinto a Jupyter, concretamente en PyCharm. Allí se llevaría a cabo un nuevo bloque del proyecto, centrado en analizar vídeos mediante el modelo de emociones entrenado. Para facilitar el traslado del modelo, la carpeta exportada fue comprimida en un archivo .zip.

Este paso fue esencial para integrar el modelo dentro del sistema de análisis en tiempo real que se desarrolló posteriormente en un entorno local.

Creación del video

Para comprobar el funcionamiento del modelo en condiciones más cercanas al mundo real, se decidió crear un vídeo de prueba compuesto por escenas seleccionadas de películas. El objetivo era evaluar si el sistema de reconocimiento emocional era capaz de detectar emociones en situaciones reales, con actores en movimiento, diferentes condiciones de luz y variedad de expresiones faciales.

Se priorizaron escenas donde apareciera únicamente una persona en pantalla, preferiblemente de frente, ya que esto facilitaba la detección de rostros por parte del modelo. También se valoraron aspectos como la calidad de la actuación, el nivel de iluminación, la claridad de la expresión facial y la estabilidad de la cámara.

Las escenas fueron seleccionadas manualmente de películas que ya se encontraban disponibles en mi ordenador. Se utilizó la aplicación OBS Studio para grabar la pantalla mientras se reproducían dichas escenas. Esto permitió capturar exactamente los fragmentos deseados sin necesidad de manipular directamente los archivos de vídeo originales.

Una vez recopiladas todas las grabaciones, el montaje final se realizó con el editor de vídeo iMovie. En este entorno se ajustó la duración de cada escena, se eliminaron fragmentos innecesarios y se ensamblaron todas las partes en un único archivo de vídeo. El resultado final fue exportado y transferido al entorno de desarrollo PyCharm, donde se integró dentro del sistema de análisis emocional en tiempo real.

Este vídeo personalizado sirvió como banco de pruebas para evaluar el rendimiento del modelo fuera del entorno controlado del dataset, y aportó un componente creativo y experimental al proyecto.

Prueba del modelo

Figura 23 Codigo webcam 1

```
modelo = TFSMLayer("modelo_pelicula3", call_endpoint="serving_default")

emociones = ['Enfado', 'Feliz', 'Miedo', 'Triste', 'Sorpresa', 'Neutral']

colores = {
    'Enfado': (0, 0, 255),           # Rojo
    'Feliz': (0, 215, 255),          # Amarillo
    'Miedo': (128, 0, 128),          # Morado
    'Triste': (180, 130, 70),        # Azul
    'Sorpresa': (0, 165, 255),       # Naranja
    'Neutral': (160, 160, 160)       # Gris
}

# Inicializar cámara
cap = cv2.VideoCapture(0)

# Inicializar MediaPipe face detection
mp_face = mp.solutions.face_detection
face_detection = mp_face.FaceDetection(model_selection=0, min_detection_confidence=0.5)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    resultado = face_detection.process(rgb)

    if resultado.detections:
        for deteccion in resultado.detections:
            bbox = deteccion.location_data.relative_bounding_box
            h, w, _ = frame.shape
            x = int(bbox.xmin * w)
            y = int(bbox.ymin * h)
            w_box = int(bbox.width * w)
            h_box = int(bbox.height * h)

            x, y = max(x, 0), max(y, 0)
            w_box, h_box = min(w_box, frame.shape[1] - x), min(h_box, frame.shape[0] - y)

            rostro = frame[y:y + h_box, x:x + w_box]
```

Figura 24 Código webcam 2

```
            rostro = frame[y:y + h_box, x:x + w_box]

            try:
                rostro = cv2.cvtColor(rostro, cv2.COLOR_BGR2GRAY)
                rostro = cv2.resize(rostro, (96, 96))
                rostro = rostro.astype("float32") / 255.0
                rostro = np.expand_dims(rostro, axis=-1)
                rostro = np.expand_dims(rostro, axis=0)

                salida = modelo(rostro)
                clave_salida = list(salida.keys())[0]
                salida_np = salida[clave_salida].numpy()[0]

                idx = np.argmax(salida_np)
                emocion = emociones[idx]

                texto = f"{emocion}"
                color = colores.get(emocion, (255, 255, 255))

                cv2.rectangle(frame, (x, y), (x + w_box, y + h_box), color, 2)
                cv2.rectangle(frame, (x, y - 35), (x + w_box, y), color, -1)

                cv2.putText(frame, texto, (x + 5, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
                            0.8, (0, 0, 0), 2, cv2.LINE_AA)
                cv2.putText(frame, texto, (x + 5, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
                            0.8, (255, 255, 255), 1, cv2.LINE_AA)

            except Exception as e:
                print("Error al procesar rostro:", e)

    cv2.imshow("Detección de Emociones", frame)

    if cv2.waitKey(1) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()
face_detection.close()
```

Este código corresponde a la fase final del proyecto, en la que se prueba el modelo de reconocimiento de emociones en tiempo real utilizando la webcam del ordenador. El objetivo principal de esta etapa es comprobar el comportamiento del modelo en condiciones reales de uso, es decir, con rostros en movimiento, variaciones de iluminación y expresiones naturales.

El script utiliza OpenCV para capturar vídeo desde la cámara y MediaPipe para detectar automáticamente el rostro en cada fotograma. Una vez detectado, el rostro se recorta, se convierte a escala de grises y se redimensiona al tamaño esperado por el modelo (96x96). Además, se normaliza y se ajustan las dimensiones del array para que sea compatible con la entrada del modelo.

El modelo cargado, ya entrenado previamente, se ejecuta sobre cada rostro detectado. A partir de la salida del modelo, se obtiene la emoción con mayor probabilidad y se muestra sobre la imagen en tiempo real. Para mejorar la presentación, cada emoción se dibuja con un color distinto (por ejemplo, rojo para "Enfadado", azul para "Triste", amarillo para "Feliz", etc.). También se dibujan rectángulos alrededor del rostro para identificar la zona procesada.

El código incluye medidas de seguridad como el control de errores en caso de que el rostro no pueda procesarse, y permite cerrar la ventana pulsando la tecla Esc. Este sistema fue útil no solo para comprobar la robustez del modelo ante condiciones no controladas, sino también para visualizar en directo cómo responde el sistema ante diferentes expresiones faciales.

Cabe destacar que, aunque el rendimiento cuantitativo del modelo sobre el conjunto de test no fue muy elevado, su comportamiento en vídeo en tiempo real fue razonablemente coherente, especialmente con emociones más marcadas como "Feliz" o "Enfadado". Esta fase resultó clave para valorar la viabilidad práctica del modelo y confirmar que, a pesar de sus limitaciones, es capaz de reconocer emociones con cierta estabilidad en situaciones reales.

Figura 25 Felicidad

Figura 26 Neutral

Figura 27 Sorpresa

Figura 28 Tristeza

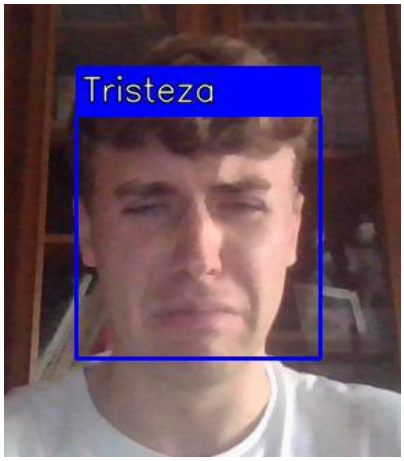
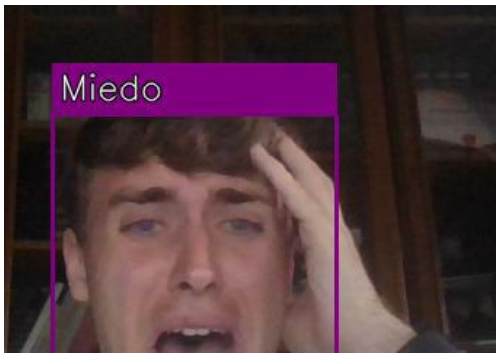


Figura 29 Enfado

Figura 30 Miedo



Código final. Análisis de emociones en un vídeo externo

Figura 31 Código vídeo 1

```
video_path = "Videos/Peliculaaa.mp4"
csv_path = "emociones_video.csv"
output_path = "Videos/Video.Emociones.mp4"

emociones = ['Enfado', 'Feliz', 'Miedo', 'Triste', 'Sorpresa', 'Neutral']
colores = {
    'Enfado': (0, 0, 255),          # Rojo
    'Feliz': (0, 215, 255),         # Amarillo
    'Miedo': (128, 0, 128),         # Morado
    'Triste': (180, 130, 70),       # Azul
    'Sorpresa': (0, 165, 255),      # Naranja
    'Neutral': (160, 160, 160)      # Gris
}


modelo = TFSMLayer("modelo_pelicula3", call_endpoint="serving_default")
video = cv2.VideoCapture(video_path)
fps = int(video.get(cv2.CAP_PROP_FPS))
width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

# CSV
csv_file = open(csv_path, "w", newline="", encoding="utf-8")
csv_writer = csv.writer(csv_file)
csv_writer.writerow(["Segundo", "Emocion", "Confianza"])

frame_num = 0
emocion_anterior = "Neutral"
confianza_anterior = 0.0
ultimo_cambio = -10

# Mediapipe
mp_face = mp.solutions.face_detection
face_detection = mp_face.FaceDetection(model_selection=0, min_detection_confidence=0.5)

while True:
    ret, frame = video.read()
```

Figura 32 Código vídeo 2

```
while True:
    ret, frame = video.read()
    if not ret:
        break

    segundo_actual = frame_num // fps
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    resultado = face_detection.process(rgb)

    if resultado.detections:
        deteccion = max(resultado.detections, key=lambda d: d.score[0])
        bbox = deteccion.location_data.relative_bounding_box
        x = int(bbox.xmin * width)
        y = int(bbox.ymin * height)
        w = int(bbox.width * width)
        h = int(bbox.height * height)
        x, y = max(x, 0), max(y, 0)
        w, h = min(w, width - x), min(h, height - y)

        rostro = frame[y:y + h, x:x + w]
        try:
            rostro = cv2.cvtColor(rostro, cv2.COLOR_BGR2GRAY)
            rostro = cv2.resize(rostro, (96, 96))
            rostro = rostro.astype("float32") / 255.0
            rostro = np.expand_dims(rostro, axis=-1)
            rostro = np.expand_dims(rostro, axis=0)

            salida = modelo(rostro)
            clave_salida = list(salida.keys())[0]

            salida_np = salida[clave_salida].numpy()[0]

            idx = np.argmax(salida_np)
            emocion = emociones[idx]
            confianza = float(salida_np[idx])

            if confianza >= 0.2 and (frame_num - ultimo_cambio > fps * 2):
                emocion_anterior = emocion
                confianza_anterior = confianza
                ultimo_cambio = frame_num
```

Figura 33 Código vídeo 3

```
            if confianza >= 0.2 and (frame_num - ultimo_cambio > fps * 2):
                emocion_anterior = emocion
                confianza_anterior = confianza
                ultimo_cambio = frame_num

            if frame_num - ultimo_cambio > fps * 2:
                emocion_anterior = emocion
                confianza_anterior = confianza
                ultimo_cambio = frame_num

            color = colores.get(emocion_anterior, (255, 255, 255))
            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
            cv2.rectangle(frame, (x, y - 30), (x + w, y), color, -1)
            cv2.putText(frame, emocion_anterior, (x + 5, y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)

            csv_writer.writerow([segundo_actual, emocion_anterior, round(confianza_anterior, 3)])

        except Exception as e:
            print(f" Error en el frame {frame_num} con detección: {e}")

    else:
        print(f" No se detectaron rostros en el frame {frame_num}")

    out.write(frame)
    frame_num += 1

video.release()
out.release()
csv_file.close()
face_detection.close()
print(" Análisis completado con éxito.")
```

Una vez validado el modelo de emociones en tiempo real mediante webcam, se procedió a su aplicación sobre el vídeo cinematográfico previamente creado. El objetivo era analizar, fotograma a fotograma, qué emociones se detectaban en los rostros de los actores a lo largo del vídeo, y registrar esos resultados tanto visualmente como en un archivo CSV.

Este proceso se dividió en varias etapas clave:

Preparación de recursos y modelo:

Se cargó el modelo previamente exportado en formato SavedModel, y se definieron las rutas tanto para el vídeo de entrada como para el archivo CSV de salida y el nuevo vídeo generado con anotaciones emocionales. También se

definió la lista de emociones posibles y se asignó a cada una un color personalizado para su visualización sobre los rostros detectados.

Configuración del vídeo:

Mediante OpenCV, se cargó el vídeo y se extrajeron propiedades esenciales como el número de fotogramas por segundo (fps), el ancho y la altura. A partir de estos datos, se configuró un objeto VideoWriter para poder guardar el vídeo resultante con las emociones superpuestas.

Inicialización del CSV:

Se creó un archivo CSV donde se fue registrando, para cada segundo del vídeo, la emoción detectada y el nivel de confianza asociado. La cabecera del archivo incluye las columnas: "Segundo", "Emoción" y "Confianza".

Detección de rostros con Mediapipe:

Para cada fotograma se aplicó la herramienta FaceDetection de Mediapipe. En caso de haber múltiples detecciones, se seleccionaba únicamente la de mayor puntuación para evitar inconsistencias en escenas con varios rostros.

Preprocesamiento y predicción:

El rostro detectado se recortaba, convertía a escala de grises, redimensionaba a 96x96 píxeles, normalizaba y transformaba al formato adecuado para la entrada del modelo. A continuación, se obtenía la predicción del modelo, seleccionando la emoción con mayor probabilidad y extrayendo su nivel de confianza.

Visualización y exportación:

Se introdujo un sistema para evitar cambios bruscos entre emociones, permitiendo actualizar la emoción mostrada solo si han transcurrido al menos dos segundos desde el último cambio y la confianza supera un umbral mínimo (0.2). Esta emoción se representaba gráficamente sobre el rostro con un rectángulo de color y su nombre correspondiente. La emoción y su confianza también se almacenaban en el CSV asociadas al segundo correspondiente del vídeo.

Gestión de errores y finalización:

Se incluyeron bloques try-except para capturar posibles errores durante el procesamiento de los rostros, y mensajes informativos si no se detectaba ningún rostro en un fotograma determinado. Al finalizar el análisis completo del vídeo, se liberaban todos los recursos abiertos como el vídeo, el archivo CSV y la instancia de Mediapipe.

Este proceso permitió generar un vídeo anotado emocionalmente, con resultados más estables visualmente gracias al nuevo control temporal entre cambios de emoción, y una base de datos estructurada para posteriores análisis cuantitativos.

Enlace al video

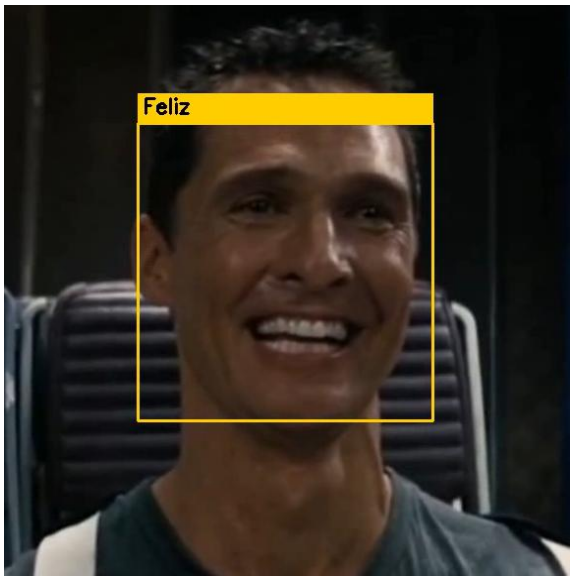
https://youtu.be/08tG8oguknM

Discusión

El sistema desarrollado en este trabajo ha permitido construir un flujo completo para el reconocimiento automático de emociones faciales, desde el entrenamiento del modelo hasta su aplicación práctica en vídeos reales. No obstante, aunque los resultados obtenidos son funcionales y demuestran una clara capacidad del modelo para detectar emociones básicas, también deben ser interpretados dentro de los límites y desafíos propios del proyecto.

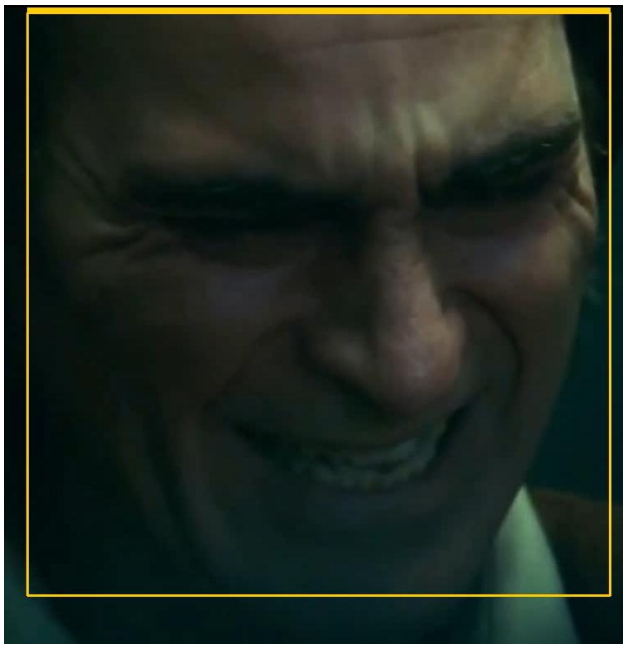
Uno de los aspectos clave a destacar es la ambigüedad emocional presente en ciertas escenas. Las redes neuronales convolucionales que se han utilizado analizan únicamente las expresiones faciales visibles en una imagen fija o en un fotograma, sin tener en cuenta el contexto, la narrativa, ni las emociones internas del personaje. Esto representa una limitación estructural del sistema: el modelo no puede comprender la intención emocional real más allá de la expresión visible.

Figura 34 Ejemplo 1



Un ejemplo representativo es una de las escenas extraídas de la película Interestelar, en la que el protagonista comienza riendo al ver un vídeo familiar, pero poco después se descompone emocionalmente y rompe a llorar. A pesar de que el trasfondo emocional real es la tristeza, el modelo detecta correctamente una sonrisa en el rostro y, por tanto, clasifica el momento como "felicidad". Esta disonancia entre emoción expresada y emoción sentida refleja una limitación que no se puede resolver únicamente con visión por computadora.

Figura 35 Ejemplo2



Del mismo modo, en una escena del *Joker*, el personaje se encuentra emocionalmente devastado, pero debido a su trastorno neurológico, se ríe incontrolablemente. El modelo interpreta esa expresión como "feliz", aunque el contexto psicológico indique todo lo contrario. Estos casos reflejan que el sistema, al estar limitado a una entrada visual aislada, no puede acceder a la complejidad emocional real del ser humano.

Otro aspecto crítico del proyecto ha sido el uso del dataset FER-2013. Aunque este dataset es una referencia común en el ámbito del reconocimiento de emociones, su aplicación directa a entornos más dinámicos y variados, como un vídeo con escenas reales, demostró ser insuficiente. Las clases estaban desbalanceadas, la iluminación era homogénea, y las expresiones eran más artificiales que naturales. Esto afectaba al rendimiento del modelo cuando se le pedía analizar escenas con iluminación difícil, ángulos laterales o emociones sutiles.

Para solventarlo, se optó por ampliar manualmente el dataset mediante la creación de un vídeo casero. Este vídeo, grabado y etiquetado personalmente, aportó ejemplos adicionales más variados, con diferentes condiciones de iluminación y expresiones reales. Este paso resultó clave para adaptar mejor el

modelo a las exigencias del vídeo final y mejorar su generalización. Sin embargo, también introdujo nuevos desafíos, como la necesidad de renombrar y reorganizar las carpetas, unificarlas en idioma y formato, y lidiar con errores durante la preparación del dataset (por ejemplo, imágenes perdidas o carpetas vacías).

Finalmente, otro elemento discutible ha sido la variabilidad de la emoción a lo largo del tiempo en un vídeo. Para evitar predicciones inestables frame a frame, se introdujo una lógica que fija la emoción detectada durante al menos dos segundos antes de permitir una actualización. Esta decisión ayudó a estabilizar la salida del modelo y evitar cambios erráticos, pero también puede enmascarar transiciones rápidas entre emociones reales.

En resumen, aunque el sistema logra detectar correctamente muchas emociones y su comportamiento es estable en condiciones controladas, su precisión disminuye en situaciones complejas donde el rostro no refleja la verdadera emoción interna. Además, el rendimiento del modelo depende en gran medida de la calidad y representatividad del dataset, siendo necesaria una ampliación o recolección personalizada para aplicaciones en vídeos reales.

Estos límites no disminuyen el valor del proyecto, sino que abren líneas claras de mejora: integración de información contextual, combinación con análisis de voz o texto, uso de arquitecturas multimodales y entrenamiento con datasets más ricos emocionalmente. Este trabajo no representa el fin, sino una base sólida sobre la cual seguir desarrollando y perfeccionando el modelo.

Conclusiones

A lo largo de este Trabajo Fin de Grado se ha desarrollado un sistema de reconocimiento automático de emociones faciales utilizando técnicas de aprendizaje profundo, con el objetivo de analizar emociones en imágenes y vídeo de forma eficaz. El proyecto ha implicado un proceso completo de ingeniería de datos, desde la recolección y preparación del dataset hasta la evaluación de un modelo funcional aplicado sobre un vídeo real.

Inicialmente, se trabajó exclusivamente con el dataset FER-2013, pero los resultados preliminares mostraron limitaciones en la precisión del modelo. Esta

situación motivó la creación de un vídeo casero del que se extrajeron nuevas imágenes, con el fin de ampliar y equilibrar el conjunto de datos. Esta decisión permitió mejorar la diversidad del dataset y adaptar mejor el modelo a situaciones reales.

El uso de transferencia de aprendizaje con MobileNetV2 resultó una estrategia eficaz frente al entrenamiento desde cero, dadas las limitaciones de tamaño y equilibrio del dataset original. El modelo fue evaluado mediante métricas como *accuracy*, matriz de confusión y *F1-score*, evidenciando una mayor capacidad de predicción en clases como “Feliz” o “Neutral”, pero con margen de mejora en emociones menos representadas como “Triste” o “Miedo”. La visualización de curvas de aprendizaje y la clasificación por clase permitieron identificar estas debilidades de forma precisa.

La parte más destacable del proyecto fue la integración del modelo en una aplicación práctica: el análisis *frame a frame* de un vídeo cinematográfico, mostrando las emociones detectadas en pantalla en tiempo real, con anotaciones visuales sobre los rostros y exportando un registro estructurado en formato CSV. Para evitar parpadeos rápidos entre clases, se incorporó un mecanismo de estabilización temporal que solo actualiza la emoción si han transcurrido al menos dos segundos desde el último cambio y si la confianza supera un umbral mínimo. Esta fase permitió comprobar visualmente el rendimiento del sistema y demostró el potencial de este tipo de modelos en aplicaciones de análisis emocional audiovisual.

En definitiva, se ha conseguido implementar un sistema funcional de reconocimiento de emociones, integrando todas las fases necesarias desde el preprocesamiento hasta el despliegue final.

Este trabajo, además de su valor técnico, ha servido como una experiencia completa en el desarrollo y evaluación de sistemas inteligentes aplicados al reconocimiento emocional, con implicaciones tanto técnicas como éticas relevantes.

Cosas que he aprendido

Este Trabajo Fin de Grado ha supuesto una oportunidad única para consolidar y ampliar mis conocimientos en diversas áreas de la ciencia de datos, inteligencia artificial y procesamiento audiovisual. A lo largo del desarrollo del proyecto, he aprendido y puesto en práctica los siguientes aspectos:

Funcionamiento de redes neuronales profundas

He comprendido en profundidad cómo se estructura y entrena una red neuronal convolucional (CNN) para tareas de clasificación de imágenes, así como el papel de las capas, funciones de activación, normalización y técnicas como el dropout para mejorar la generalización.

Transferencia de aprendizaje

He aprendido a reutilizar modelos preentrenados como MobileNetV2 para adaptarlos a nuevos conjuntos de datos, comprendiendo cómo congelar capas, ajustar pesos y personalizar la arquitectura final para tareas específicas.

Preprocesamiento de imágenes

He trabajado con técnicas fundamentales para adaptar datos de entrada a modelos de deep learning, incluyendo escalado de grises, normalización, redimensionamiento y formato de tensores.

Curación y expansión de datasets

He reforzado la importancia del equilibrio entre clases en el entrenamiento, aprendiendo a extraer y etiquetar imágenes a partir de vídeos caseros y clips cinematográficos, para aumentar la representatividad del conjunto de datos.

Evaluación de modelos de clasificación

He utilizado métricas como accuracy, matriz de confusión y F1-score para evaluar el rendimiento del modelo, aprendiendo a interpretar los resultados por clase y a detectar problemas como el sobreajuste o el desbalanceo.

Uso de herramientas especializadas

Mediapipe: para la detección de rostros de forma rápida y eficiente en imágenes y vídeos.

OpenCV: para leer, procesar, editar y guardar vídeos, además de superponer información visual como textos, colores y recuadros.

TensorFlow: para el manejo de modelos preentrenados y exportación en formato SavedModel.

Google Colab y PyCharm: como entornos de trabajo para prototipado rápido, pruebas y desarrollo local.

Trabajo autónomo y resolución de problemas

A lo largo del proyecto me he enfrentado a numerosos desafíos técnicos (problemas de compatibilidad, limitaciones del dataset, comportamiento errático del modelo en vídeo, etc.), lo que me ha obligado a buscar soluciones, experimentar, comparar enfoques y validar hipótesis de forma crítica.

Comunicación y documentación del proyecto

He aprendido a estructurar y presentar un trabajo técnico de forma clara, justificando las decisiones tomadas, explicando los resultados obtenidos y elaborando documentación comprensible para terceros.

Contribución a los Objetivos de Desarrollo Sostenible (ODS)

Aunque este proyecto se enmarca principalmente en el ámbito tecnológico, su aplicación y desarrollo tiene implicaciones que pueden alinearse con varios de los Objetivos de Desarrollo Sostenible definidos por la ONU para la Agenda 2030. A continuación, se detallan los ODS con los que guarda mayor relación:

ODS 3: Salud y Bienestar

El reconocimiento automático de emociones tiene un enorme potencial en el ámbito de la salud mental. Este tipo de herramientas podrían integrarse en aplicaciones destinadas a:

- Detectar patrones de emociones negativas persistentes (como tristeza o ira) en personas con riesgo de depresión o ansiedad.
- Ofrecer asistencia emocional no invasiva en entornos como centros educativos, hospitales o servicios de atención al cliente.
  En este sentido, el modelo desarrollado es un paso hacia sistemas que puedan monitorear el estado emocional y proporcionar alertas o recomendaciones preventivas.

ODS 4: Educación de calidad

El proyecto implica un fuerte componente educativo y de autoaprendizaje:

- Se han utilizado herramientas accesibles (Python, Jupyter, PyCharm, TensorFlow, etc.) que forman parte de la educación técnica moderna.
- La elaboración del modelo y todo el proceso de mejora progresiva refuerzan competencias clave en IA, ciencia de datos y pensamiento crítico.
  Además, este tipo de tecnología podría aplicarse en entornos educativos inclusivos, ayudando a detectar emociones en el aula o adaptar los contenidos al estado emocional del estudiante.

ODS 9: Industria, Innovación e Infraestructura

El proyecto utiliza y promueve tecnologías innovadoras en el campo de la visión por computador y la inteligencia artificial:

- Se ha implementado un modelo funcional capaz de analizar emociones en vídeo en tiempo real.
- La integración entre datasets, preprocesamiento, entrenamiento con data augmentation y despliegue del modelo representa una infraestructura de IA ligera y reutilizable.

Esto se alinea con el objetivo de fomentar la innovación y el acceso a soluciones tecnológicas avanzadas, incluso con recursos limitados.

ODS 10: Reducción de las desigualdades

Aunque de forma indirecta, el proyecto también contribuye a este objetivo:

- El análisis emocional automático puede facilitar la comunicación de personas con trastornos del habla o dificultades para expresar sus emociones verbalmente.
- Permitir el acceso a estas tecnologías en entornos vulnerables (por ejemplo, con modelos que funcionen en dispositivos de bajo coste gracias a MobileNet) puede ayudar a reducir barreras de accesibilidad tecnológica.

Reflexión Final: Ética y Sostenibilidad en la Inteligencia Artificial Emocional

Este proyecto no solo ha sido una experiencia técnica y creativa, sino también un ejercicio de reflexión sobre el papel de la inteligencia artificial en la comprensión de las emociones humanas. A lo largo del desarrollo, se ha evidenciado que, aunque los modelos como el utilizado pueden alcanzar buenos niveles de precisión en tareas visuales, la emoción humana es compleja, ambigua y profundamente contextual. Intentar reducirla a una categoría a partir de una sola imagen puede ser útil, pero también peligroso si no se interpreta con cuidado.

Además, se ha procurado mantener el proyecto en una línea ética y sostenible: se ha trabajado con datos generados o procesados personalmente, sin recopilar rostros de terceros sin consentimiento, y se ha reutilizado equipamiento propio y software libre para evitar costes energéticos y materiales innecesarios. Esta es una demostración de que es posible avanzar tecnológicamente sin perder de vista los principios éticos y los Objetivos de Desarrollo Sostenible.

El camino hacia una inteligencia artificial verdaderamente empática no depende solo de avances en modelos más complejos, sino también de cómo decidimos usarla, a quién afecta, con qué fin y bajo qué valores. Este proyecto es una pequeña contribución a ese debate, con la convicción de que el desarrollo tecnológico debe ir siempre acompañado de responsabilidad humana.

Repositorio GitHub

https://github.com/luisllobell/TFG.git

Referencias

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (pp. 265–283). https://www.tensorflow.org/

Apple Inc. (n.d.). *iMovie – Apple (ES)*. Recuperado el 1 de mayo de 2025 de https://www.apple.com/es/imovie/

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. https://opencv.org/

Chowdary, M. K., Nguyen, T. N., & Hemanth, D. J. (2021). Deep learning-based facial emotion recognition for human–computer interaction applications. *Neural Computing and Applications, 35*, 23311–23328. https://doi.org/10.1007/s00521-021-06012-8

GeeksforGeeks. (2025). *Emotion detection using convolutional neural networks (CNNs)*. Recuperado el 28 de abril de 2025 de https://www.geeksforgeeks.org/emotion-detection-using-convolutional-neural-networks-cnns/

JetBrains. (n.d.). *PyCharm – The Python IDE for Professional Developers*. Recuperado el 1 de mayo de 2025 de https://www.jetbrains.com/pycharm/

Kaggle. (n.d.). *FER-2013 facial expression recognition dataset*. Recuperado el 2 de mayo de 2025 de https://www.kaggle.com/datasets/msambare/fer2013

Matplotlib Developers. (2025). *Matplotlib Documentation*. https://matplotlib.org/stable/contents.html

MDPI. (2019). Facial expression recognition: A survey. *Symmetry, 11*(10), 1189. https://www.mdpi.com/2073-8994/11/10/1189

MDPI. (2021). Facial expression recognition using transfer learning and deep CNNs. *Electronics, 10*(9), 1036. https://www.mdpi.com/2079-9292/10/9/1036

Nature. (2023). Multi-source transfer learning for facial emotion recognition. *Scientific Reports*. https://www.nature.com/articles/s41598-023-48250-x

Nature. (2024). An Efficient MobileNetV2-based architecture for facial expression recognition. *Scientific Reports*. https://www.nature.com/articles/s41598-024-58736-x

NumPy Developers. (2025). *NumPy User Guide*. https://numpy.org/doc/

OBS Studio Contributors. (n.d.). *OBS Studio – Open Broadcaster Software*. Recuperado el 1 de mayo de 2025 de https://obsproject.com/

OpenAI. (2024). *ChatGPT (April 2024 version)*. https://chat.openai.com/

OpenCV. (n.d.). *Face detection using Haar cascades*. OpenCV Documentation. Recuperado el 2 de mayo de 2025 de https://docs.opencv.org/4.x/d7/d00/tutorial_meanshift.html

TensorFlow. (n.d.). *Keras API documentation*. https://www.tensorflow.org/api_docs/python/tf/keras

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace. https://docs.python.org/3/

Wikipedia contributors. (n.d.). *Emotion recognition*. Wikipedia. Recuperado el 28 de abril de 2025 de https://en.wikipedia.org/wiki/Emotion_recognition