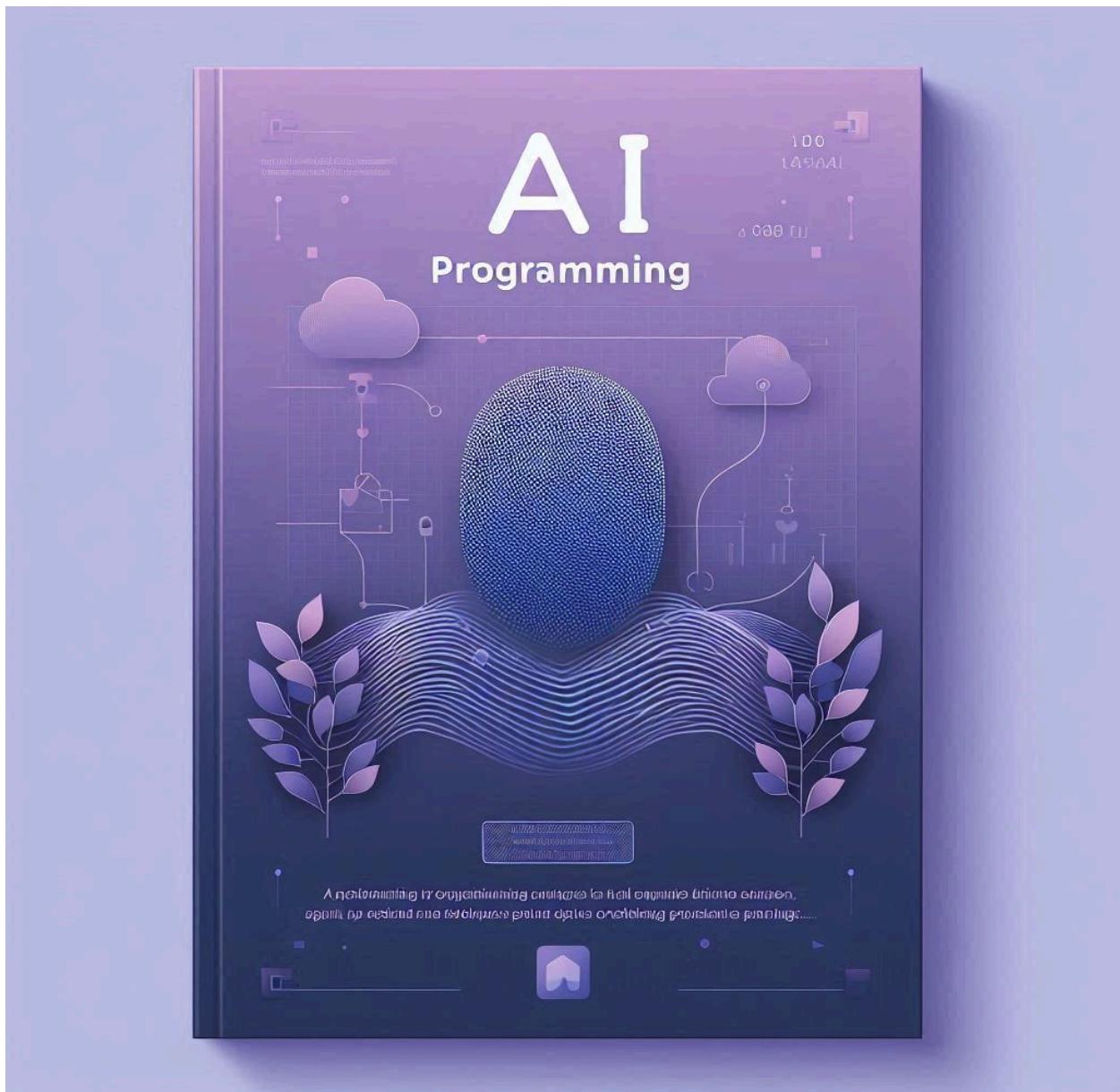


Una histor-IA detrás de cada imagen



Autors: Robert Lopez Machuca/Laura Fornós Ramírez

Curs: Especialització IA i Big Data

MP6. Projecte d' intel·ligència artificial i Big Data

Index

1. Introducción	3
2. Tecnologías utilizadas y decisiones de diseño	4
2.1 Qué lenguajes, frameworks, librerías o herramientas se han utilizado.	4
2.2 ¿Por qué se han elegido esas tecnologías?	6
2.3 Organización del trabajo entre los miembros del grupo.	7
3. Creación y procesamiento del dataset	10
3.1 Cómo se ha recopilado el conjunto de imágenes.	10
3.2 Proceso de etiquetado manual (estructura del archivo de etiquetas, herramientas utilizadas si aplica).	11
3.3 Técnicas de preprocesamiento aplicadas (cambios de tamaño, normalización, filtrado, etc.)	12
4. Descripción de los clasificadores y su entrenamiento	15
4.1 Qué clasificadores se han creado.	15
4.2 Tipo de modelo utilizado en cada uno (KNN, SVM, redes neuronales, etc.).	16
4.3 Hiperparámetros elegidos y motivación.	17
4.4 Dificultades encontradas durante el entrenamiento.	18
5. Resultados y matrices de confusión	19
5.1 Resultados numéricos obtenidos por cada clasificador.	19
Resumen rápido:	19
5.2 Matrices de confusión individuales.	25
5.3 Análisis comparativo entre clasificadores.	31
5.4 Reflexión sobre los que funcionan mejor/peor y posibles mejoras.	37
6. Integración del modelo LLM y ejemplos generados	40
6.1 Cómo se construyen los prompts a partir de los resultados.	40
6.2 Qué modelo de lenguaje se ha utilizado (API externa o modelo local).	41
6.3 Ejemplos de textos generados.	42
6.4 Valoración de los resultados y posibles mejoras.	43
7. Descripción de la aplicación web	44
7.1 Cómo está estructurada la aplicación (frontend/backend).	44
7.2 Flujo de funcionamiento desde la subida de imagen hasta la generación del texto.	45
8. Conclusiones y reflexión final	47
8.1 Qué se ha aprendido en el desarrollo del proyecto.	47
8.2 Qué partes han sido más difíciles o interesantes.	47
8.3 Qué harías diferente si volvieras a empezar	48
9. Bibliografía	49
9.1 Lista de Imágenes	49
9.2 Lista de Links de Páginas Web utilizadas para el Web Scraping	50
9.3 Instalaciones en VS Code	51

1. Introducción

El objetivo de este proyecto es desarrollar una aplicación web capaz de analizar la imagen de una persona y generar automáticamente una descripción junto a una pequeña historia inventada a partir de las características detectadas. Para lograrlo hemos entrenado clasificadores que extraigan información visual de la imagen, y posteriormente integrar esos resultados con un modelo de lenguaje (LLM) que genere el texto de forma automática.

En primer lugar usamos Web-Scraping mediante Selenium en Python para poder descargar y almacenar miles de imágenes correspondientes a 19 subcategorías en 6 categorías (Edad, Sexo, Pelo, Gafas, Profesión, Emociones).

Nuestro proyecto cuenta con 6 clasificadores entrenados usando redes neuronales CNN y la base de datos de las imágenes preprocesadas, los cuales corresponden a 6 categorías que nos proporcionan cada uno una matriz de confusión, un reporte de clasificaciones y la precisión de una validación.

Todo esto se combina en una API hecha mediante Flask para crear una conexión remota mediante IP, la cual especificamos en nuestro HTML.

Nuestra interfaz web en la cual insertar la imagen para crear la historia, consiste de una zona de drop y una interfaz con 6 cajas de resultado y una extra que nos devuelve una historia con las etiquetas que nos han proporcionado los clasificadores.

Todo esto se ha logrado usando Visual Studio Code para la parte de programación en Python, Html y CSS, junto con la aplicación de Google Drive para control y distribución de las versiones y mejoras del programa entre los dos integrantes del grupo.

2. Tecnologías utilizadas y decisiones de diseño

2.1 Qué lenguajes, frameworks, librerías o herramientas se han utilizado.

Lenguajes principales

- **Python**: Lenguaje principal utilizado en el desarrollo del backend, el preprocesamiento de imágenes y la implementación de modelos de deep learning.
- **HTML5**: Utilizado para la estructura del frontend, permitiendo una presentación clara y accesible al usuario.
- **CSS3**: Aplicado para el diseño visual del frontend, incluyendo estilos responsivos, personalizados y uso de variables CSS para mantenibilidad.
- **JavaScript (nativo)**: Lenguaje que dota de interactividad a la interfaz web, maneja eventos (como arrastrar y soltar imágenes), envía archivos mediante fetch y actualiza dinámicamente la interfaz.

Frameworks y librerías

Backend (Python)

- **Flask**: Framework web ligero que permite exponer los modelos como una API RESTful accesible desde el frontend.
- **Flask-CORS**: Permite la interoperabilidad entre dominios, solucionando restricciones CORS y permitiendo que el frontend se comunique con el backend.
- **TensorFlow / Keras**: Utilizadas para crear, entrenar, guardar y cargar redes neuronales convolucionales (CNN).
- **OpenCV (cv2)**: Herramienta esencial para el preprocesamiento de imágenes, como la detección de rostros, cambio de color y redimensionamiento.
- **NumPy**: Para la manipulación eficiente de arrays y operaciones numéricas con imágenes.
- **Pillow (PIL)**: Para manejar imágenes en formato RGB y convertir entre formatos.

- **rembg**: Se utiliza para eliminar automáticamente el fondo de las imágenes, facilitando el enfoque en el sujeto principal.
- **seaborn**: Para crear visualizaciones estadísticas, especialmente para representar matrices de confusión con anotaciones más claras.
- **scikit-learn**: confusion_matrix, classification_report: Para evaluar el rendimiento del modelo mediante métricas estándar de clasificación.
- **ImageDataGenerator (Keras)**: Para la generación de datos aumentados (rotación, escalado, zoom, etc.) que mejoran la robustez del modelo durante el entrenamiento.
- **EarlyStopping y ModelCheckpoint (Keras callbacks)**: Ayudan a evitar el sobreajuste durante el entrenamiento y a guardar el mejor modelo automáticamente.
- **modeLR**: Módulo personalizado (local) que consiste en las redes neuronales CNN.

Frontend (HTML/CSS/JS)

- **HTML5**: Estructura la interfaz de usuario, con formularios, botones y zonas de visualización.
- **CSS3**: Define estilos modernos y adaptables a diferentes dispositivos (responsive design).
- **Google Fonts (Poppins)**: Tipografía moderna y clara que mejora la experiencia visual del usuario.
- **JavaScript (nativo)**: Maneja eventos en la página, comunica con el backend y actualiza el contenido sin recargar la página.
- **OpenAI API**: Se integra un modelo de lenguaje natural (LLM) para generar descripciones creativas y contextuales a partir de las predicciones generadas por los modelos CNN. Esto añade una capa de interpretación semántica y narrativa a los resultados.

2.2 ¿Por qué se han elegido esas tecnologías?

Backend

El uso de **Python** se justifica por su versatilidad y fuerte ecosistema en inteligencia artificial y procesamiento de imágenes. El framework **Flask** fue seleccionado por su simplicidad y ligereza, lo que resulta adecuado para una API cuyo objetivo es recibir una imagen y devolver predicciones de manera rápida y eficiente.

Las bibliotecas **TensorFlow** y **Keras** permiten una integración fluida de modelos de deep learning previamente entrenados. **OpenCV** y **NumPy** proporcionan herramientas robustas y optimizadas para el tratamiento y transformación de imágenes, una etapa crítica en el pipeline del sistema.

La integración de la API de **OpenAI** permite enriquecer la experiencia del usuario generando descripciones narrativas o historias relacionadas con la imagen procesada. Esta funcionalidad va más allá de una simple clasificación, ofreciendo una interpretación más contextual y natural del resultado. El uso de **Flask-CORS** resuelve los posibles problemas de seguridad y acceso entre dominios al permitir que la interfaz web pueda comunicarse correctamente con la API incluso si se alojan en servidores distintos.

Frontend

En el frontend, el uso de **HTML5** y **CSS3** permite construir una interfaz moderna, accesible y adaptable a distintos dispositivos. El uso de variables CSS facilita el mantenimiento y la personalización visual de la página. **Google Fonts** mejora la legibilidad y la estética general.

El uso de **JavaScript** permite una interacción fluida con el usuario sin necesidad de recargar la página. Mediante el uso de fetch, los archivos de imagen se envían a la API de forma asíncrona, y los resultados se presentan de manera dinámica, mejorando significativamente la experiencia de uso.

2.3 Organización del trabajo entre los miembros del grupo.

En nuestro proyecto primero hicimos una plantilla de trabajo con Trello y decidimos que al principio se dividiría por las categorías de imágenes, las cuales fueron:

Robert

Edad: Niño, Joven, Adulto, Anciano,

Emociones: Feliz, Serio, Triste, Enfadado, Gafas: Con Gafas, Sin gafas

Pelo: Pelo corto, Pelo largo.

Laura

Sexo: Hombre, Mujer

Profesiones: Bombero, Policía, Camarero, Médico

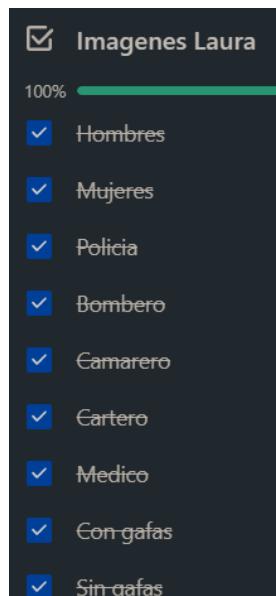


Imagen 1

Imagen 2

Imagen 3

También, decidimos dividirlo por tareas de programación tal que así: Robert se encargaría de los preprocesos y limpieza de imágenes y Laura se encargaría de crear la base del proyecto, las carpetas, que subió a Drive, ya que decidimos que sería la mejor herramienta para compartirnos las partes del proyecto.

El proyecto base se veía tal que así, aunque no teníamos en si programado nada, simplemente era el concepto de los archivos que queríamos y como podríamos hacer el proyecto:

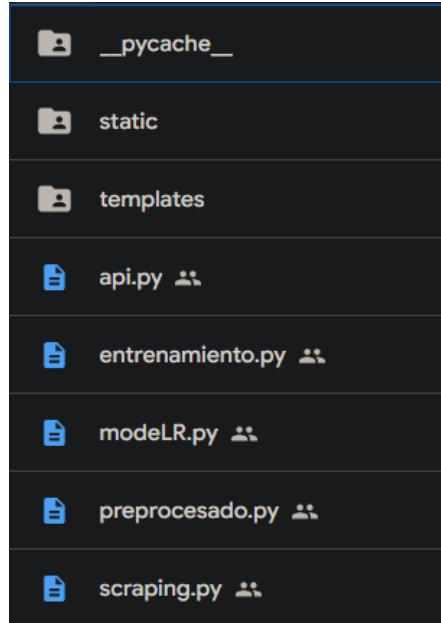


Imagen 4

Después, cada uno decidió hacer su archivo de entrenamiento para sus categorías para poder cada entrenamiento tener su carpeta de imágenes asociada y sus etiquetas correctamente, para poder tener un control de las instalaciones de librerías requeridas, Laura creó un archivo de ‘Bibliografía’ donde íbamos añadiendo tanto las instalaciones en Visual Studio Code como links de interés. Y sobretodo decidimos hacer el código de las CNN en un archivo que es modeLR.py, así evitamos tocar las redes neuronales si no es necesario.

Los códigos de entrenamientos así quedaron:

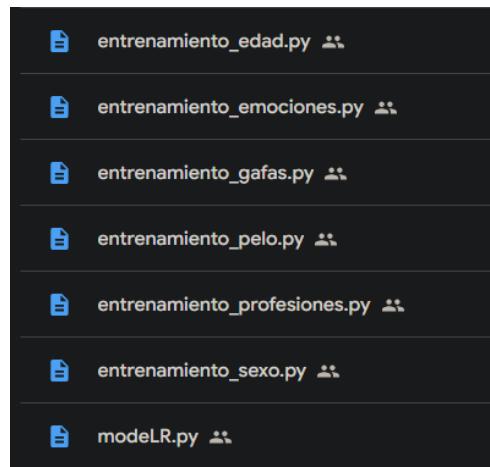
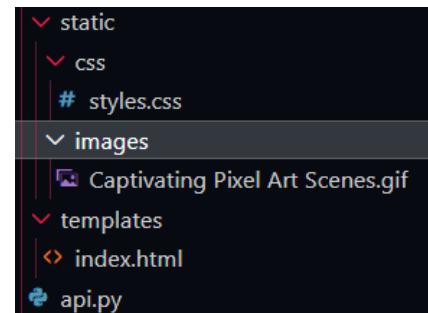


Imagen 5

Mas adelante, Laura se encargó de hacer el código del HTML junto con los estilos CSS y Robert creó el archivo la API, para poder hacer este último añadimos un pequeño tutorial para realizar la conexión en el archivo ‘Bibliografía’.



de

Imagen 6

Por último, nos dividimos también por archivos, las pruebas que íbamos a realizar con cada modelo para comprobar su funcionamiento, antes de hacer estas pruebas o futuras modificaciones a los códigos, decidimos dejar una copia del programa completo como ‘backup’ por prevención. El proyecto final quedó dividido en una parte de documentación (carpetas y archivos que necesitamos) que hizo Laura y la parte de programa que fue hecha mediante los dos juntando los entrenamientos y modelos que adquirimos. Así se vería el resultado final:

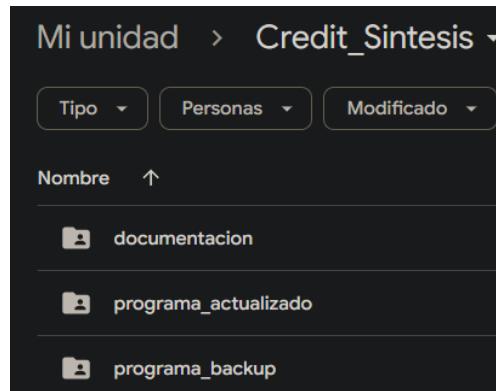
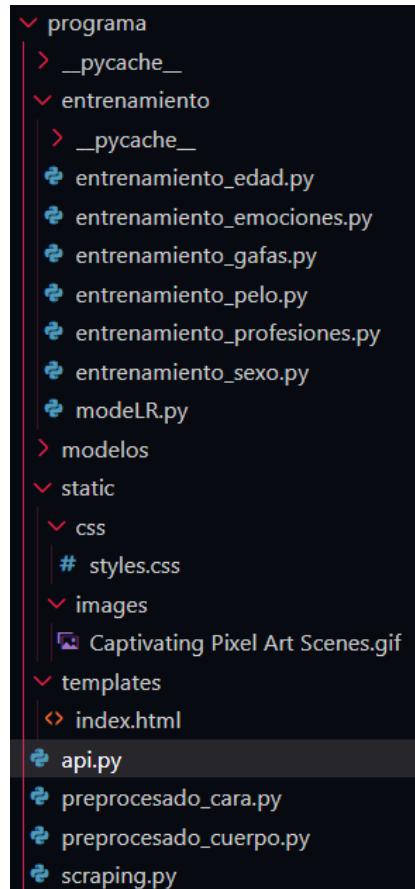
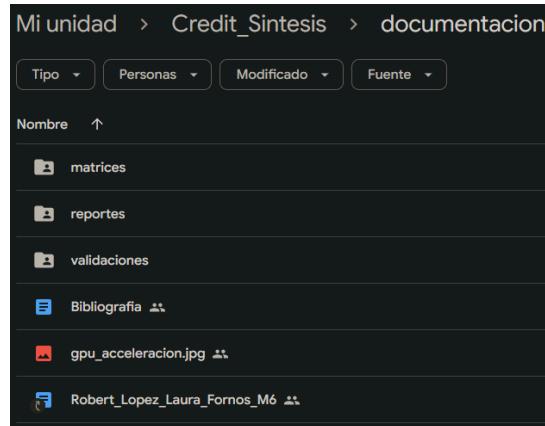


Imagen 7

Imagen 8

Imagen 9



3. Creación y procesamiento del dataset

3.1 Cómo se ha recopilado el conjunto de imágenes.

Hemos recopilado nuestro conjunto de imágenes mediante web scraping automatizado desde varias páginas web, utilizando Python con las librerías Selenium y Requests. Aunque el programa no está limitado a una temática: está diseñado para funcionar con cualquier categoría de imágenes, simplemente cambiando el enlace base de búsqueda. En la [bibliografía](#) incluimos los enlaces utilizados para cada categoría.

El archivo se llama Scrapy.py para poder ver el código completo que se explica a continuación.

Utilizamos **Selenium con Google Chrome**, configurado con webdriver_manager, para automatizar la navegación por la web. También definimos una carpeta de destino en el escritorio donde se almacenan automáticamente todas las imágenes descargadas, organizadas con nombres secuenciales.

Partimos de una **URL de búsqueda** en páginas web de imágenes de stock y fotografías, centrándonos en la palabra ‘Portrait’ ya que esta nos ayuda a buscar ya sea policía, mujer, niño, etc. de torso hacia la cabeza.

Implementamos un sistema de **scroll automático** que se repite 20 veces por página para asegurarnos de que la web cargue todas las imágenes posibles, emulando el comportamiento de un usuario real.

Una vez cargadas las imágenes, buscamos los elementos y extraemos sus URLs desde el atributo src.

Antes de descargar, aplicamos varios filtros:

- Descartamos imágenes codificadas como data: image (base64).
- Evitamos duplicados usando un **hash MD5** de cada URL.
- Ignoramos imágenes demasiado pequeñas (menos de 224x200 / 224 x 224 píxeles), ya que no serían útiles para el entrenamiento del modelo.

Las imágenes que cumplen los requisitos se descargan con requests y se verifican con **PIL (Pillow)**. Luego las guardamos en la carpeta de destino con nombres como policia_00001.jpg, policia_00002.jpg, etc.

Establecemos dos límites para controlar la cantidad de datos:

- Un máximo de **3000 imágenes** por ejecución.
- O un máximo de **100 páginas** recorridas, lo que ocurra primero.

Ventajas de usar este scraping:

- Tenemos control total sobre el **tipo y la calidad** de las imágenes descargadas.
- El proceso es **automático**, lo que mejora la eficiencia.
- Filtramos imágenes irrelevantes o de baja calidad, evitando ruido visual.
- Es una base sólida para entrenar modelos de clasificación de imágenes.
 -

3.2 Proceso de etiquetado manual (estructura del archivo de etiquetas, herramientas utilizadas si aplica).

Cuando se descargan las imágenes, por ejemplo unas 3000 imágenes de cada profesión o 6000 imágenes de hombres, se hace el siguiente proceso:

La imagen se guarda con el nombre de la categoría que es ‘policía’ junto con el número de imagen que es ‘0001’ quedaría tal que así: ‘policia_0001.jpg’

En el código se descargan las imágenes de una subcategoría, es decir , ponemos un link que descarga una imagen y la guardará en una carpeta llamada como la categoría a la que pertenece: una imagen de policía se guardará como ‘policia_0001.jpg’ en la carpeta ‘policía’ que se guarda en el escrito o carpeta que deseemos.

De esta manera, cuando se descargan las x imágenes que queramos, tendremos una carpeta con imágenes etiquetadas, esta carpeta será la que usaremos para apuntar nuestro código de preprocesamiento para procesar las imágenes.

```
# Ruta de destino
carpeta_destino = os.path.join(os.path.expanduser("~/Desktop", "policia"))
os.makedirs(carpeta_destino, exist_ok=True)
```

Imagen 10

```
nombre_archivo = os.path.join(carpeta_destino, f"policia_{descargadas:05d}.jpg")
with open(nombre_archivo, "wb") as f:
    f.write(img_data)
```

Imagen 11

3.3 Técnicas de preprocesamiento aplicadas (cambios de tamaño, normalización, filtrado, etc.)

El preprocesamiento es clave para garantizar que todas las imágenes tengan el mismo formato, sean comparables y estén centradas en la información relevante. A continuación se explican los pasos aplicados antes de usar las imágenes para entrenar o predecir con los modelos:

Eliminación del fondo (rembg)

Se usa la librería **rembg** para eliminar el fondo de cada imagen y conservar únicamente el objeto principal (la persona). Esto permite centrarse en el cuerpo o el rostro, eliminando distracciones del entorno.

Detección del cuerpo o del rostro

Para el preprocessado de profesiones: Se localiza el **contorno más grande** en la imagen binaria para detectar la **silueta del cuerpo completo**, ya que la ropa o herramientas visibles son clave para esta clasificación.

Para el resto de modelos: Se utiliza el **clasificador Haar** (**haarcascade_frontalface_default.xml**) para detectar automáticamente el **rostro**.

Una vez detectado el cuerpo o el rostro:

Se **recorta la región** donde se encontró. Se añade un **margin (padding)** alrededor para evitar recortes bruscos de partes importantes (como el pelo, mentón u hombros)

Redimensionado (resize)

Las imágenes se escalan a un tamaño uniforme de **224x224 píxeles**, requisito común en redes neuronales convolucionales. Esto garantiza una entrada estandarizada al modelo.

Luego usamos padding cuando el recorte no es exactamente cuadrado:

Se calcula cuánto espacio falta por cada lado (arriba, abajo, izquierda, derecha). Se **añade relleno blanco (RGB: 255,255,255)** para completar hasta **224x224**, manteniendo la proporción original. Evita deformaciones al escalar y permite que el objeto (rostro o cuerpo) esté **centrado** en la imagen.

Procesamiento paralelo (aceleración del preprocessamiento)

Se aprovechan todos los núcleos del procesador utilizando **ProcessPoolExecutor** para procesar múltiples imágenes al mismo tiempo. Esto reduce considerablemente el tiempo total de procesamiento, especialmente útil cuando el conjunto de datos contiene cientos o miles de imágenes.

Conversión de formatos de imagen

La imagen original se convierte desde bytes a formato PIL (RGB) para poder trabajar con ella de forma más flexible. Luego se convierte de nuevo a **formato OpenCV (BGR)** porque es el formato que necesita OpenCV para realizar operaciones como la detección de rostros.

Manejo de errores y control de imágenes inválidas

Si una imagen no contiene rostro (o cuerpo, en su caso), se **descarta automáticamente** del conjunto de datos. Esto evita que se incluyan imágenes no válidas en el entrenamiento, lo cual podría afectar negativamente al rendimiento del modelo.

Reutilización de imágenes ya procesadas

Si una imagen ya fue procesada previamente y está guardada en la carpeta de salida, no se vuelve a procesar (a menos que se active el modo force_preprocess). Esto **ahorra tiempo** en ejecuciones repetidas

Beneficios del preprocessamiento aplicado:

- Elimina **ruido** como fondos irrelevantes.
- Centra la atención en **rostros o cuerpos**, que contienen la información relevante.
- **Mejora el rendimiento** de los modelos al entregar entradas limpias y normalizadas.

4. Descripción de los clasificadores y su entrenamiento

4.1 Qué clasificadores se han creado.

En este proyecto hemos creado **seis clasificadores distintos** ([ver imagen 5](#)), cada uno entrenado para predecir una característica visual concreta, sin interferirse entre ellos. En esos código solo se encuentra el entrenamiento, las redes neuronales se encuentran en otro que es el de modeLR. Estas son las categorías y funcionamiento general:

1. **Edad** (niño, joven, adulto, anciano) **Emociones** (feliz, serio, triste, enfadado)
Profesiones (policía, bombero, camarero, médico) → 4 clases
2. **Gafas** (con gafas / sin gafas) **Sexo** (hombre / mujer) **Pelo** (corto / largo) → 2 clases



4.2 Tipo de modelo utilizado en cada uno (KNN, SVM, redes neuronales, etc.).

Para todos los clasificadores hemos utilizado un modelo de **Red Neuronal Convolucional (CNN)**. Las CNN son un tipo de red neuronal especialmente eficaz para trabajar con **imágenes**. Funcionan identificando patrones en las imágenes, como bordes, formas o texturas, y combinándolos para reconocer objetos o características más complejas.

¿Cómo funcionan las CNN (de forma sencilla)?

Conv2D → BatchNormalization → MaxPooling2D

×4 bloques

↓

Flatten → Dense(256) → Dropout(0.5) → Dense(salida)

Convoluciones (Conv2D): Son como filtros que se mueven por la imagen para detectar patrones (bordes, formas, texturas). A medida que se añaden más capas, la red aprende a reconocer cosas más complejas.

Normalización (BatchNormalization): Acelera y estabiliza el entrenamiento ajustando automáticamente la activación de las neuronas.

Reducción de tamaño (MaxPooling): Reduce la imagen a la mitad en cada bloque, manteniendo lo más importante. Esto permite que el modelo sea más rápido y se enfoque en las partes clave.

Aplanamiento (Flatten): Convierte la imagen procesada (que ahora es una especie de mapa de características) en un vector para poder pasarlo a la parte final.

Clasificación (Dense 256): Conecta todas las neuronas para tomar la decisión final: ¿a qué clase pertenece la imagen? Se usa **softmax** porque queremos que la salida sea una probabilidad para cada clase.

Dropout(0.5): apaga aleatoriamente un 50% de las conexiones durante el entrenamiento para evitar que el modelo dependa demasiado de ciertos valores (previene sobreajuste).

Salida: es la última capa, con tantas neuronas como clases tenga el clasificador. Utiliza softmax para dar una probabilidad a cada clase.

4.3 Hiperparámetros elegidos y motivación.

Estos han sido los hiperparametros que hemos usado tanto en el código de modeLR.py y el de entrenamiento_x.py.

img_size = (224, 224): Tamaño al que se redimensionan todas las imágenes. Esto asegura que todas tengan la misma forma y que coincidan con la entrada de la red neuronal.

batch_size = 32: Número de imágenes que se procesan juntas antes de actualizar los pesos.. Un valor moderado como 32 permite entrenar rápido sin consumir demasiada memoria.

Epochs = 10: Número máximo de veces que el modelo va a pasar por todos los datos de entrenamiento. Gracias a EarlyStopping, puede parar antes si deja de mejorar.

learning_rate = 1e-4 (o 0.0001): Indica la velocidad a la que el modelo ajusta sus pesos. Un valor bajo como este permite aprender de forma estable y sin saltos bruscos.

EarlyStopping: Detiene el entrenamiento si el modelo deja de mejorar durante varias épocas. En este caso, se detiene si no mejora en 2 épocas seguidas (patience=2), y recupera los mejores pesos anteriores (restore_best_weights=True).

Dropout(0.5): Apaga aleatoriamente el 50% de las neuronas durante el entrenamiento. Esto evita que el modelo se sobreentrene (overfitting) y ayuda a generalizar mejor.

4.4 Dificultades encontradas durante el entrenamiento.

Aunque el entrenamiento ha funcionado bien, hemos tenido que aplicar ciertas precauciones para evitar errores comunes:

- **Overfitting:** es cuando el modelo aprende demasiado bien los datos de entrenamiento y falla con nuevos datos. Para evitarlo, usé Dropout y EarlyStopping.
- **Separación modular:** decidí crear la arquitectura del modelo en un archivo separado (modeLR.py) para facilitar pruebas, cambios y mantenimiento sin tocar todos los scripts.
- **Validación de clases:** incluye una verificación para asegurarnos de que el número de clases sea el correcto en cada clasificador. Esto previene errores silenciosos.

A nivel de modelos, los mayores problemas surgieron en las tres categorías donde tenemos más clases (emociones, profesiones y edades), esto debido a que hemos tenido que tener en cuenta los siguientes puntos para que funcionaran correctamente:

- Tener en cuenta la cantidad de imágenes que hay en cada subcategoría (emociones → feliz), manteniendo un equilibrio en cuanto a tener, por ejemplo, 3000 imágenes de médicos y 2500 de bomberos, están desequilibradas, pero no es una diferencia muy elevada.
- Comprobar en la matriz de confusión cuales son las categorías que crean conflicto, para poder saber que carpetas revisar si hay alguna imagen que ha sido mal preprocesada o no hay una diferencia clara entre subcategorías.
- Hicimos pruebas con data generation más robusta y resultó darnos peores resultados, así que hicimos una de solo reescalada de imágenes, ya que si rotamos imágenes, añadimos zooms, etc. nos perjudicaba muchísimo llegando a tener modelos como emociones con un 35% de accuracy.

5. Resultados y matrices de confusión

5.1 Resultados numéricos obtenidos por cada clasificador.

Ahora bien, veremos los resultados que hemos obtenido en los reportes de clasificación y las validaciones que hemos hecho, siempre del 20% de los datos, analizaremos los resultados de los siguientes parámetros:

Resumen rápido:

Métrica	¿Qué mide?	¿Cuándo importa?
Accuracy	Qué tan seguido acierta	En general
Precisión	Qué tan confiables son los positivos	Cuando los falsos positivos son malos
Recall	Qué tan bien encuentra todos los positivos	Cuando no quieras dejar pasar ningún positivo
F1-score	Balance entre precisión y recall	Cuando quieras un equilibrio entre ambos

Sexo

El primero del que hablaremos es el clasificador de sexo (hombres y mujeres). alcanzando una precisión del **91%** sobre 1861 muestras. La categoría '**hombres**' resalta por una fineza un poco más alta (**0.93**), mientras que la categoría '**mujeres**' tiene mejor **recall** (**0.93**), esto muestra, que detecta mejor los casos reales de este tipo. Ambas categorías tienen un **f1-score de 0.91**, marca de una buena mezcla entre firmeza y recall. Los promedios **macro** y **weighted** son iguales, confirmando que el modelo tiene un comportamiento estable y equilibrado para ambos tipos.

Reporte de Clasificación:				
	precision	recall	f1-score	support
hombres	0.93	0.89	0.91	930
mujeres	0.89	0.93	0.91	931
accuracy			0.91	1861
macro avg	0.91	0.91	0.91	1861
weighted avg	0.91	0.91	0.91	1861

Imagen 12

Gafas

A continuación, tenemos el modelo de gafas (con o sin ellas), con una precisión de acierto del **92%** sobre 1060 ejemplos mostrando un comportamiento muy equilibrado entre las dos clases. La clase 'con_gafas' tiene una exactitud alta del **0.96** aunque su recuerdo es **más bajo (0.88)** lo cual indica que aunque casi nunca se equivoca al etiquetar gafas. Aunque puede dejar escapar algunos casos reales. Por otra parte la clase 'sin_gafas' tiene un recuerdo excelente (**0.96**) encontrando casi todos los casos reales aunque con una precisión algo menor (**0.88**). Ambos clases presentan su numero **f1 similar a 0.91 y 0.92** lo que muestra un buen equilibrio general.

Reporte de Clasificación:				
	precision	recall	f1-score	support
con_gafas	0.96	0.87	0.91	530
sin_gafas	0.88	0.96	0.92	530
accuracy			0.92	1060
macro avg	0.92	0.92	0.92	1060
weighted avg	0.92	0.92	0.92	1060

Imagen 13

Profesiones

El modelo de profesiones (bombero, médico, policía, camarero) tiene una exactitud global del **88%** ante 1414 ejemplos. Mostrando un rendimiento bueno pero con cambios según la clase.

La clase **bombero** resalta por su buen trabajo, con exactitud del **0.98** y un **f1-score de 0.95**, lo que muestra que el modelo halla casi todos los casos de esta clase y apenas hace errores. En cambio, la clase camarero tiene la precisión más baja (**0.78**) aunque con un recall más alto (**0.87**), lo que puede querer decir que si bien detecta bien a los camareros verdaderos también suele confundirse con otras clases.

Las clases '**médico**' y '**policía**' tienen un comportamiento bueno con puntuaciones **f1 de 0.86** y **0.88** respectivamente, manteniendo una unidad. Las cifras de media macro y promedio pesado muestran un total de **0.88** en todos los ejemplos, lo que significa que el modelo funciona de manera pareja y sin mucho sesgo hacia ningún tipo.

Reporte de Clasificación:				
	precision	recall	f1-score	support
bombero	0.98	0.93	0.95	375
camarero	0.78	0.87	0.82	368
medico	0.86	0.86	0.86	339
policia	0.91	0.84	0.88	332
accuracy			0.88	1414
macro avg	0.88	0.88	0.88	1414
weighted avg	0.88	0.88	0.88	1414

Imagen 14

Edad

El modelo logra una precisión general del **67%** en 1485 muestras, con un rendimiento relativamente equilibrado entre clases. La clase ‘**anciano**’ destaca con una precisión y un f1-score de **0.84**, reflejando una identificación muy precisa. La clase ‘**niño**’ también tiene buenos resultados con un **f1-score de 0.70**.

Las clases ‘**joven**’ y ‘**adulto**’, sin embargo, tienen un peor desempeño, especialmente la clase ‘**adulto**’ con un **f1-score de 0.52**, reflejando una mayor confusión con otras edades. Aun así, los valores de macro y weighted average de alrededor de **0.67** indican un buen desempeño general y sin mucho sesgo.

Reporte de Clasificación:				
	precision	recall	f1-score	support
adulto	0.59	0.47	0.52	370
anciano	0.84	0.83	0.84	376
joven	0.54	0.69	0.61	369
nino	0.73	0.67	0.70	370
accuracy			0.67	1485
macro avg	0.67	0.67	0.67	1485
weighted avg	0.68	0.67	0.67	1485

Imagen 15

Emociones

En el modelo de emociones, hay un desempeño equitativo a lo largo de 1286 ejemplos. Con una precisión promedio del **62%**, aunque la desviación apunta a una precisión que varía por clase. ‘**feliz**’ es la clase con mejor **precisión, 0.86**, que también se traduce en el mejor **f1-score del modelo: 0.89**. Aun sin llegar a la perfección, el algoritmo identifica bien a la inmensa mayoría de las personas felices sin demasiados errores. En cambio, en las clases ‘enfadado’ y ‘triste’ lo hace bastante mal, por debajo de la mitad su **f1-score (0.44 en el caso de ‘triste’ y 0.50 en el caso de ‘enfadado’)**, lo que indica que muestra una tendencia clara a confundirse y no sabe muy bien separar estas emociones.

Con ‘**neutral**’ logra algo aceptable, su **f1-score es justo 0.62**. La media (macro y weighted) en torno a **0.61–0.62** nos deja ver que modelo, si bien útil, falla en acertar a ciertas emociones.

Reporte de Clasificación:				
	precision	recall	f1-score	support
enfadado	0.67	0.40	0.50	336
feliz	0.86	0.91	0.89	313
neutral	0.53	0.73	0.62	305
triste	0.45	0.44	0.44	332
accuracy			0.62	1286
macro avg	0.62	0.62	0.61	1286
weighted avg	0.62	0.62	0.61	1286

Imagen 16

Pelo

Este modelo presenta un **rendimiento sólido** con una **precisión global del 82%** sobre un total de 1656 muestras. Las dos clases ('**corto**' y '**largo**') están equilibradas en desempeño, destacando 'corto' con un **recall de 0.87**, lo que indica que el modelo detecta correctamente la mayoría de los casos con pelo corto.

Por otro lado, la clase '**largo**' logra una mayor precisión (**0.86**), lo que sugiere que, cuando predice esta clase, acierta con más frecuencia, aunque con un recall algo menor (0.79). Ambos **f1-scores se mantienen altos (0.83 y 0.82)**, reflejando una buena armonía entre precisión y recall.

Los promedios macro y ponderado de **0.82–0.83** refuerzan que el modelo se comporta de forma estable y efectiva para ambas clases.

Reporte de Clasificación:					
	precision	recall	f1-score	support	
corto	0.80	0.87	0.83	812	
largo	0.86	0.79	0.82	844	
accuracy			0.82	1656	
macro avg	0.83	0.83	0.82	1656	
weighted avg	0.83	0.82	0.82	1656	

Imagen 17

5.2 Matrices de confusión individuales.

En este punto vamos a estar viendo las matrices que hemos obtenido de nuestros clasificadores. Las matrices de confusión constan de una absoluta y otra porcentual, que se han obtenido mediante el 20% de imágenes de cada subcategoría usado para la validación.

Sexo

Aquí podemos ver las matrices de confusión del clasificador de hombres y mujeres. El modelo tiene un **alto nivel de precisión** tanto para hombres como para mujeres. Es ligeramente **más preciso clasificando mujeres (93.34%) que hombres (88.71%)**. Aquí podemos ver las matrices de confusión del clasificador de hombres y mujeres. El modelo tiene un **alto nivel de precisión** tanto para hombres como para mujeres. Es ligeramente **más preciso clasificando mujeres (93.34%) que hombres (88.71%)**.

La mayoría de los errores ocurren al **confundir hombres con mujeres** (105 casos), más que al confundir mujeres con hombres (62 casos). Así que, si nos da como resultado la etiqueta de ‘Mujer’ hay un **11.29% de probabilidades de que pueda ser hombre**, en cambio, si nos dice que es un ‘Hombre’ lo mas seguro es que lo sea, aunque tenga menos accuracy, ya que solo tiene un **6.66% de desvío**.

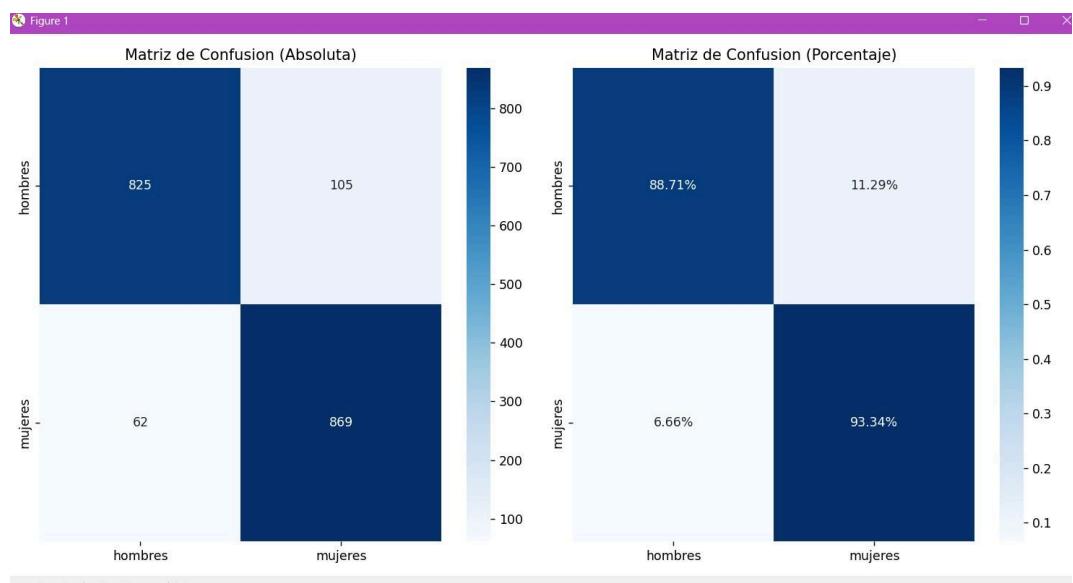


Imagen 18

Gafas

En este clasificador detectamos si la persona tiene gafas o no. El modelo tiene **muy buen rendimiento general**. Es **más preciso clasificando personas sin gafas (96.42%)** que con gafas (87.17%).

La mayoría de los errores ocurren al **confundir a personas con gafas como sin gafas** (68 casos). Solo 19 personas sin gafas fueron clasificadas erróneamente como con gafas. Así que, si nos da como resultado la etiqueta de '**con gafas**', muy probablemente lo sea ya que solo hay un **3.58%** de probabilidades de que en realidad sea una persona **sin gafas**. En cambio, si nos dice que es '**sin gafas**', lo más seguro es que realmente lo sea por la accuracy, pero hay un **12.83%** de desconfianza de que podría ser una persona **con gafas**.

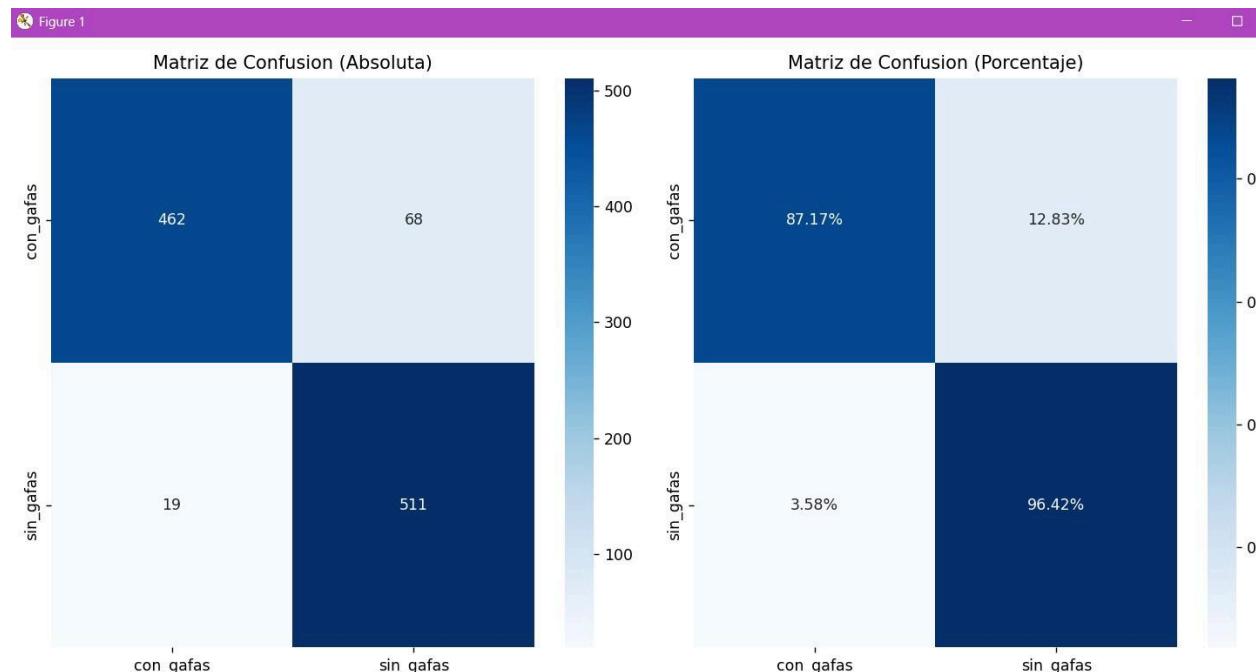


Imagen 19

Profesiones

A continuación, vemos cómo se han procesado las imágenes de profesiones. muestra un **rendimiento general bastante sólido**, con tasas de acierto superiores al **84%** en todas las categorías. La clase '**bombero**' es la mejor reconocida por el modelo, con un **93.33%** de precisión y muy poca confusión con otras profesiones, siendo la más común con '**policía**' en apenas un **3.47%**.

En cambio, las clases '**médico**' y '**policía**' presentan un poco más de ambigüedad. El modelo clasifica correctamente al **85.55%** de los médicos, pero comete errores significativos al confundirlos con **camareros** en un **13.86%** de los casos. De manera similar, los policías son identificados correctamente el **84.34%** de las veces, aunque también se los confunde principalmente con camareros (**10.24%**).

La clase '**camarero**' tiene una precisión del **87.23%**, aunque tiende a mezclarse con médicos en casi un 9% de los casos. En general, el modelo funciona bien, pero se observa que la categoría de camarero es la que más confusiones genera hacia y desde otras clases, especialmente con **médicos y policías**.

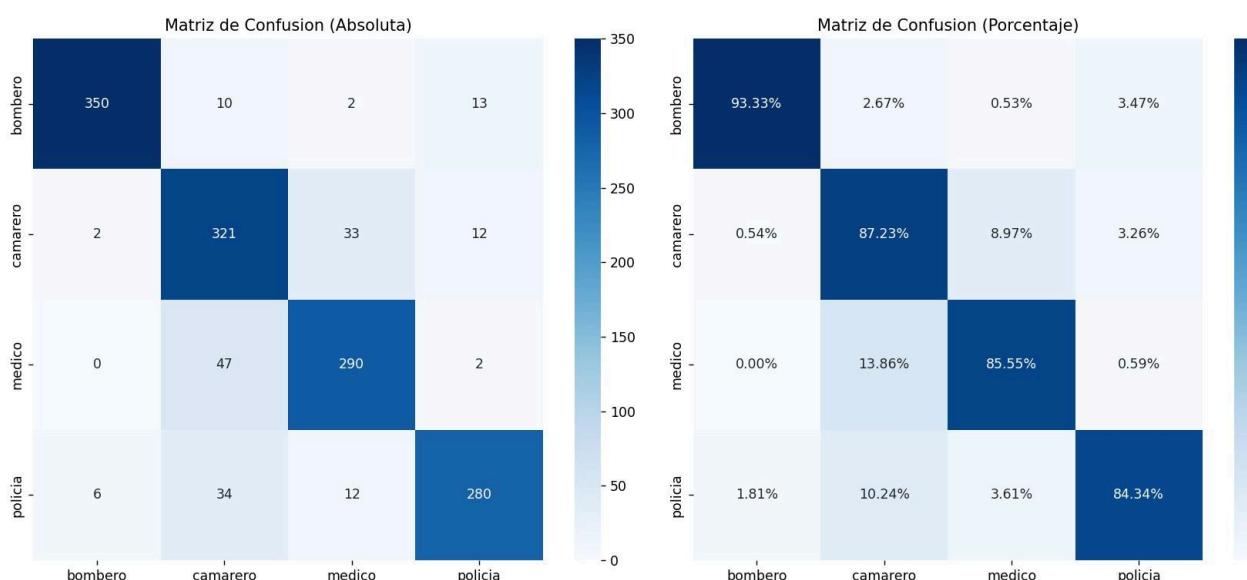


Imagen 20

Edad

Este modelo, encargado de clasificar por **rangos de edad**, muestra un desempeño desigual según la clase. La categoría '**anciano**' destaca como la mejor clasificada con un **83.24% de acierto** y muy pocos errores. En contraste, el modelo presenta importantes dificultades para identificar correctamente a los '**adultos**', con solo un **47.03%** de precisión, siendo confundidos sobre todo con '**jóvenes**' (**31.35%**) y '**ancianos**' (**14.59%**). Las categorías '**joven**' y '**niño**' obtienen resultados aceptables, con un **69.38%** y **67.30%** de precisión respectivamente, aunque ambas muestran una tendencia a confundirse entre sí.

En términos prácticos, los **errores más frecuentes** ocurren cuando se clasifica incorrectamente a un '**niño**' como '**joven**' (**102 casos**). Si el modelo predice que una persona es un '**anciano**', es muy probable que lo sea, ya que solo hay un **16.76%** de probabilidad de error. Por el contrario, si se clasifica a alguien como '**adulto**', se debe tener cautela, ya que hay un **52.97%** de probabilidades de que realmente pertenezca a otra categoría, principalmente '**joven**' o '**anciano**'.

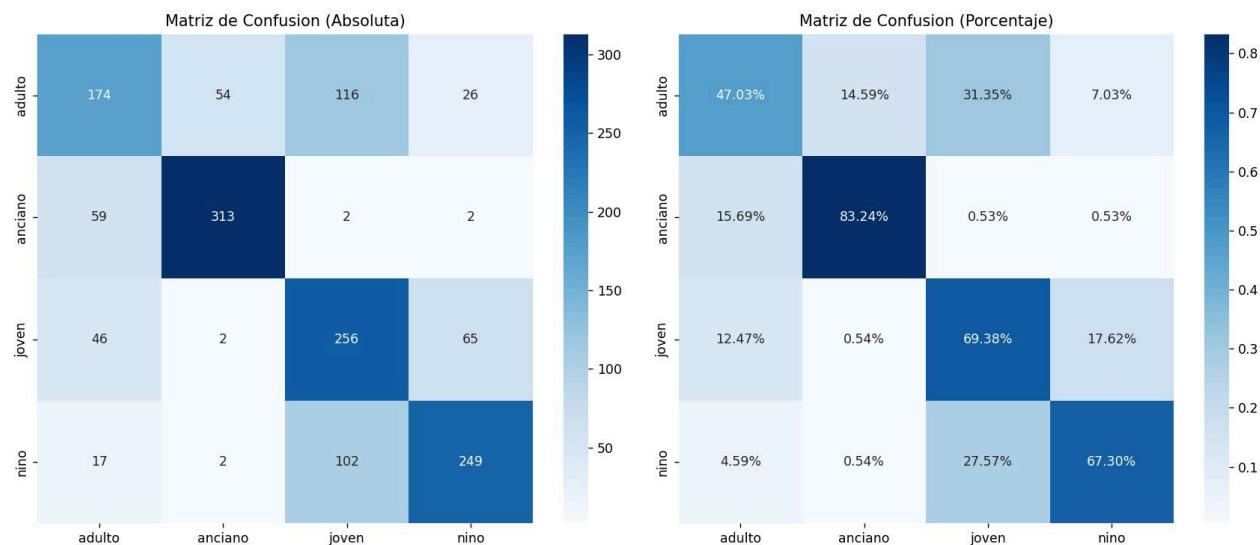


Imagen 21

Emociones

El modelo clasifica la emoción '**feliz**' con gran exactitud (**91.37%**) y la emoción 'neutral' con un rendimiento aceptable (**73.44%**), sin embargo, presenta problemas evidentes con las emociones 'triste' y especialmente 'enfadado', que solo muestra un **40.18%** de aciertos y a menudo se confunde con '**triste**' y '**neutral**'. Esto señala que el modelo puede diferenciar correctamente entre emociones positivas, pero tiene dificultades para distinguir entre emociones negativas o parecidas.

Respecto a los errores, si el modelo anticipa que una persona está '**feliz**', es muy probable que lo esté, dado que solo existe un **8.63%** de posibilidad de que no lo esté.

En cambio, si clasifica a alguien como '**enfadado**', debemos tener precaución, ya que hay un 59.82% de probabilidades de que en realidad está expresando otra emoción como 'triste' o 'neutral'.

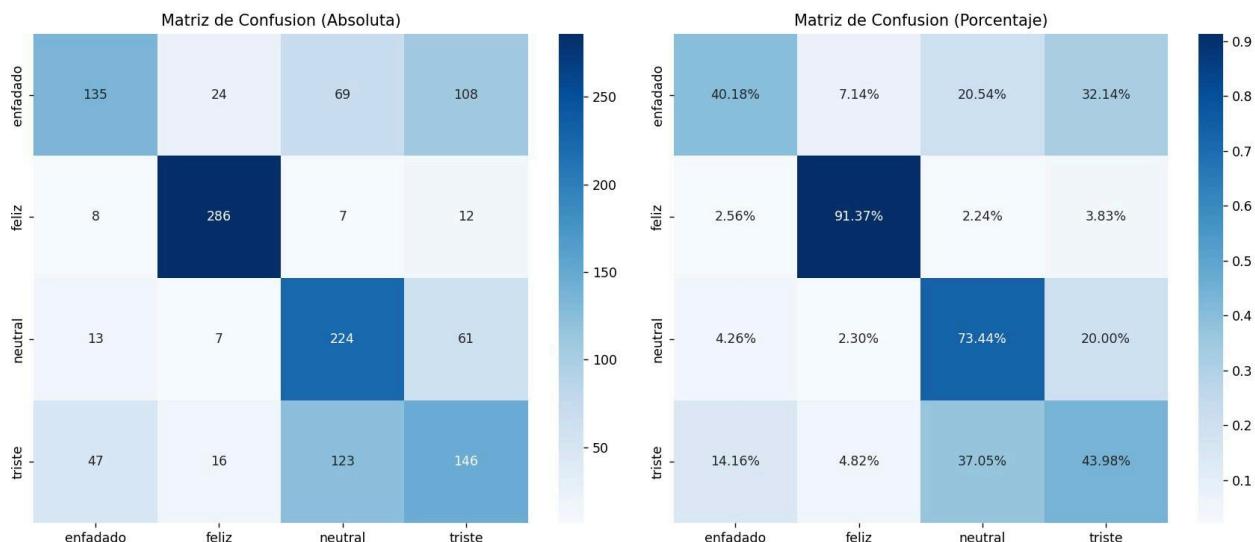


Imagen 22

Pelo

Finalmente, examinaremos el modelo de pelo **corto o largo**. Este modelo presenta un buen desempeño general, siendo más exacto al detectar personas con pelo corto (86.58%) que con pelo largo (78.55%).

La mayoría de las equivocaciones ocurren cuando se confunde a gente con pelo largo con aquellos con **pelo corto (181 casos)**, mientras que sólo **109 personas** con pelo corto fueron equivocadamente categorizados como de pelo largo. Esto sugiere que, si el modelo anticipa '**corto**', resulta bastante fiable, dado que solo un **13.42%** de esas predicciones son verdaderamente personas con pelo largo. Por otro lado, si anticipa un '**largo**', hay una mayor posibilidad de equivocación, con un **21.45%** de las situaciones correspondiendo en realidad a personas con pelo corto.

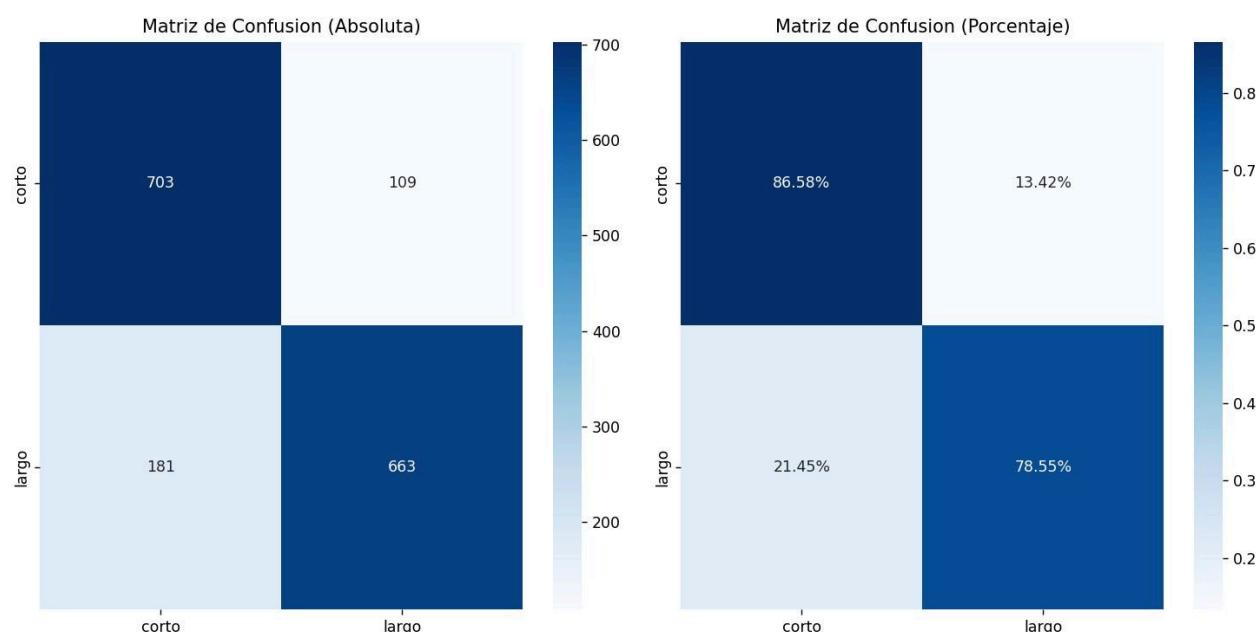


Imagen 23

5.3 Análisis comparativo entre clasificadores.

Vamos a comentar diferentes resultados y mejoras obtenidos en nuestros seis clasificadores. También expondremos cuales han sido los factores de mejora.

Sexo

El primero que veremos es el clasificador de sexo, donde no hubo muchos cambios significativos, ya que desde un primer momento las pruebas en web y los resultados de los reportes y matrices daban entorno a mas de 70% de accuracy.

Antes

Reporte de Clasificación:		precision	recall	f1-score	support
	hombres	0.94	0.78	0.85	930
	mujeres	0.81	0.95	0.88	931
accuracy				0.86	1861
macro avg		0.88	0.86	0.86	1861
weighted avg		0.88	0.86	0.86	1861

Imagen 24

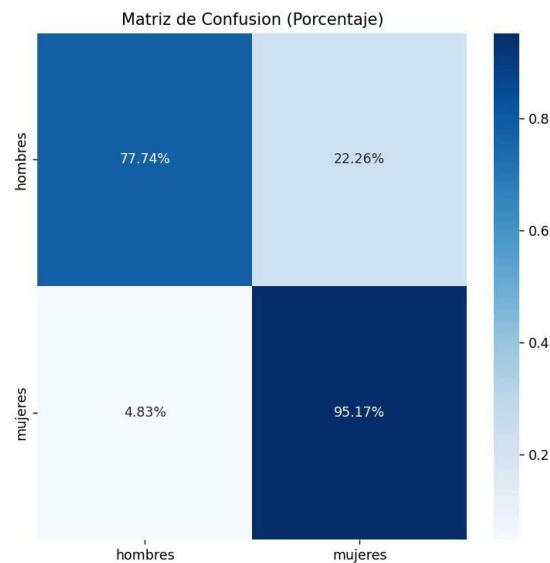


Imagen 25

Lo que hicimos para esta mejora fue usar el preprocesso de detección de rostros que encontramos en FlyPix ([link en la Bibliografía](#)), esto nos ayudó a centrar la imagen y recortar en torno al rostro, quitando ese ruido que nos provocaba confusión.

Otra de las causas fue el no implementar mujeres con /alopecia y hombres de pelo largo, cuando hicimos una revisión exhaustiva de las imágenes nos dimos cuenta de esta ausencia y añadimos unas 300 imágenes de las descripciones que he hecho a la carpeta 'Hombres' y 'Mujeres'.

Gafas

A continuación, explicaremos las mejoras en el modelo gafas, este modelo al igual que el de sexo, ha tenido un rendimiento bastante desequilibrado y no nos mejoraba del 60-70%.

Antes

	precision	recall	f1-score	support
con_gafas	0.74	0.60	0.66	530
sin_gafas	0.66	0.78	0.72	530
accuracy			0.69	1060
macro avg	0.70	0.69	0.69	1060
weighted avg	0.70	0.69	0.69	1060

Imagen 26

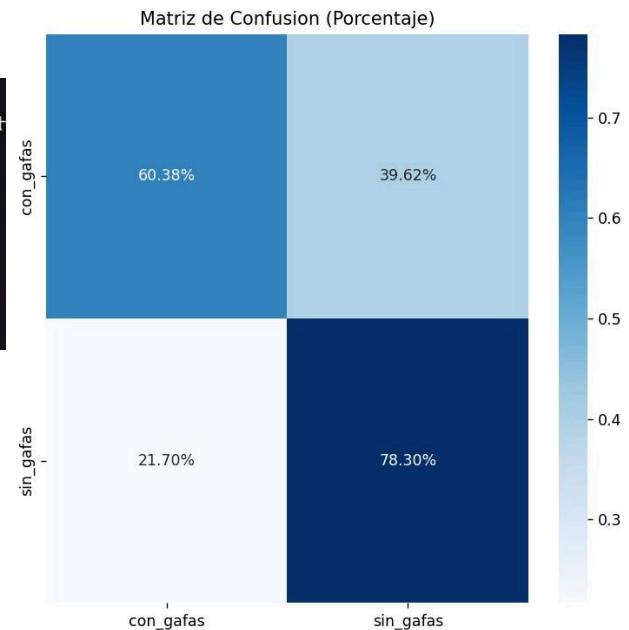


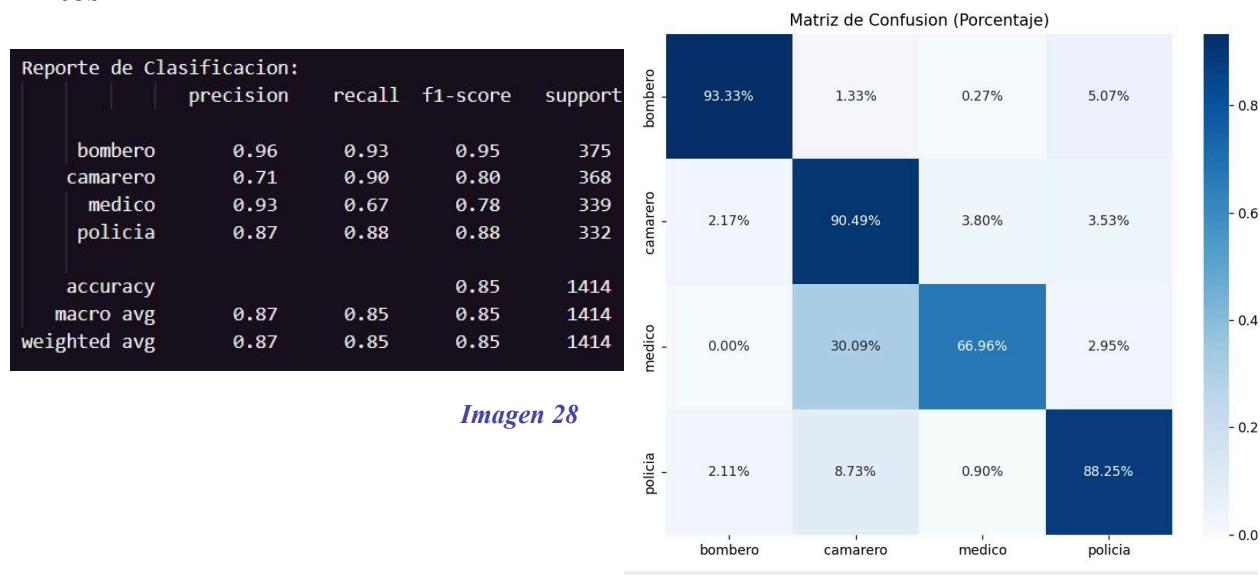
Imagen 27

Las mejoras que hicimos fue incrementar el número de neuronas y capas (antes eran dos bloques de 128 filtros) que se han incrementado, ahora cuatro bloques convolucionales con normalización por lotes, como la cantidad de filtros (de 32 hasta 256), permitiendo una extracción de características más ricas. Además, la capa densa final ha pasado a tener 256 neuronas, mejorando la capacidad del modelo para tomar decisiones más precisas tras el aplanado de los datos.

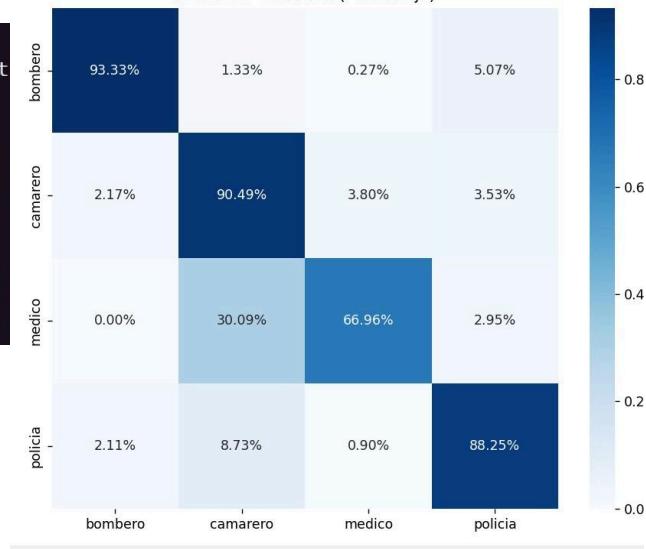
Profesiones

El siguiente que veremos es el de profesiones, el cual creemos que hicimos un gran trabajo en la mejora. Los principales puntos para ello fueron: **imágenes de mejor calidad** mediante la web ‘Getty images’ y un **preproceso único** para estas imágenes.

Antes



Matriz de Confusión (Porcentaje)



Vemos que antes había confusión entre camareros y médicos, que hablando con compañeros y revisando imágenes, creemos que era debido a que muchos camareros llevan debajo del delantal una camiseta blanca, dando pie a confundirse con las batas de médicos, lo que hicimos no solo fue reprocesar las imágenes con el detector de cuerpos, sino también hacer una limpieza de imágenes que no hubieran sido correctamente preprocesadas.

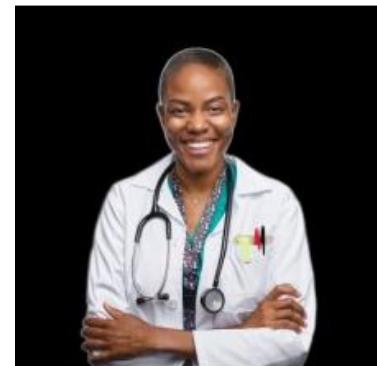


Imagen 30

Imagen 31

Edad

El siguiente es el modelo de edades, que desde un primer momento ha sido muy difícil de conseguir resultados equilibrados.

Antes

		precision	recall	f1-score	support
	adulto	0.60	0.53	0.56	370
	anciano	0.84	0.86	0.85	376
	joven	0.53	0.65	0.59	370
	nino	0.72	0.62	0.67	370
accuracy				0.67	1486
macro avg		0.67	0.67	0.67	1486
weighted avg		0.67	0.67	0.67	1486

Imagen 32

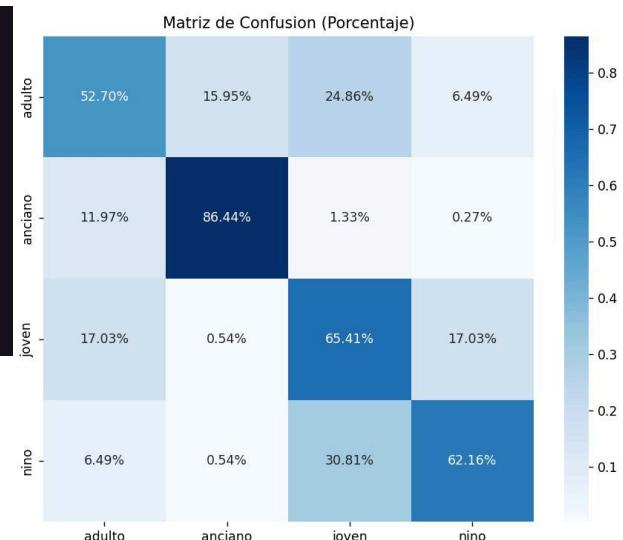


Imagen 33

Aquí vemos que antes había más desequilibrio entre las clases anciano y el resto, ya que las imágenes que encontramos no eran muy buenas y los rasgos de las personas menores a 30 se difuminaba al pre procesarlas, aun así a la práctica suele detectar bien a los mayores de edad.

No hicimos cambios en el código sino en el preprocesado donde decidimos usar menos imágenes de 3000 a 1800 en cada subcategoría, pero de calidad, de ahí a que como vimos anteriormente, se estabilizaron las clases adulto, joven y niño.

Emociones

El siguiente que veremos es el de emociones donde la emoción de felicidad siempre ha tenido un resultado muy consistente y superando los 90 puntos.

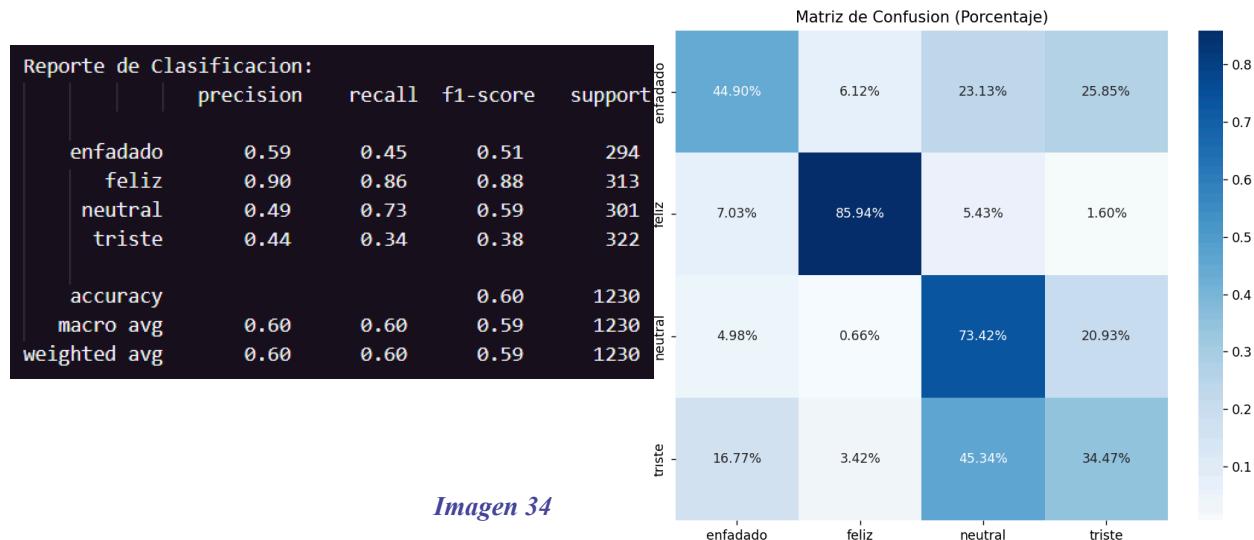


Imagen 35

La mejora principal, fue igual que en sexo, ya que usamos el recorte de rostros y nos centramos en quitar todo el ruido posible, al igual que encontramos imágenes de muy buena calidad, debido a que encontramos muchas imágenes que no reflejaban la emoción que queríamos bien.

En sí, las imágenes más asequibles de conseguir fueron las de felicidad, ya que mediante la sonrisa es para el modelo muy fácil de detectar esa emoción, en cambio el fallo antes estaba en las neutrales, enfadadas y tristes, donde como no teníamos un recorte de rostros, a veces se recorta la imagen por la mitad o con un zoom indebido.

Pelo

Por último, el modelo de pelo corto o largo, tuvo el problema

		precision	recall	f1-score	support
	corto	0.84	0.91	0.87	1064
	largo	0.84	0.71	0.77	671
		accuracy		0.84	1735
		macro avg		0.84	1735
		weighted avg		0.84	1735

Imagen 36

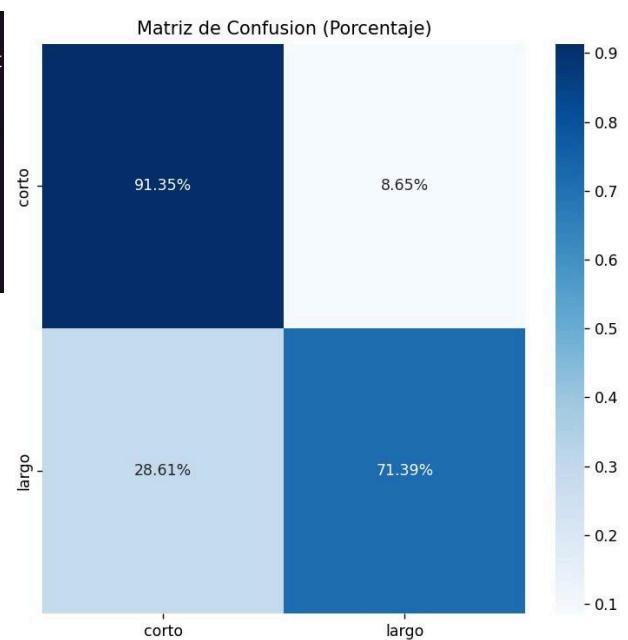


Imagen 37

El problema que encontramos en las primeras pruebas, fue que al recortar rostros con el procedimiento que explicamos anteriormente, había personas de pelo largo que perdían parte de este pelo y no se podían distinguir del corto, lo que hicimos fue buscar imágenes que fueran de torso hacia arriba o de cara más alejada, así al hacer el zoom al rostro no se desfiguraba tanto que

si era una imagen de un close-up del rostro.



Imagen 38

Imagen 39

5.4 Reflexión sobre los que funcionan mejor/peor y posibles mejoras.

✓ Nuestros Mejores Modelos

Sexo (88 % / 93 %)

Este modelo para nosotros es el mejor, tiene una diagonal muy sólida y detecta sin dificultades si es hombre o mujer, a la práctica.

Mejoras posibles:

- Podríamos refinar el recorte facial para evitar obstrucciones (gorras, pañuelos)
- Incluiríamos ejemplos más diversos (distintas etnias, edades) para reforzar la robustez.

Gafas (87 % / 96 %)

En la práctica, este modelo va muy bien, también de los mejores, suele detectar 9/10 personas con gafas y 7/10 personas sin gafas..

Mejoras posibles:

- Podríamos distinguir entre gafas más finas o de colores no muy fuertes.
- Añadiremos más ejemplos de gafas transparentes y sin filtro solar.

Profesión (diagonal muy marcada)

Salvo las confusiones entre camareros y médicos, suele detectar muy bien los uniformes y contornos, reflejándose en la matriz.

Mejoras posibles:

- Implementaremos una clasificación sin colores rgb, a lo mejor podría funcionar no depender del color.

- Dividir en un primer clasificador por si tienen uniforme o no, donde policía y bombero irían y camarero y médico a parte.

Pelo (86 % / 78 %)

Suele haber confusiones en cuanto a tipos de pelo (rizado, liso, etc) y colores, pero por lo general detecta bien la longitud del pelo.

Mejoras posibles:

- Podríamos preprocessar colores (balance de blancos, corrección gamma) para reducir la variación de iluminación.
- Recortaremos estrictamente la región del pelo y aumentaremos ejemplos de tonos menos comunes.

✗ Nuestros Peores Modelos

Emociones (diagonal débil)

Este modelo tiene bastantes confusiones entre ‘neutral’, ‘triste’, ‘enfadado’, seguramente porque estas expresiones no están muy bien definidas y en imágenes que no sean un close-up del rostro, puede tender a no detectar bien el rostro.

Mejoras posibles:

- Entrenaremos con imágenes de expresiones más exageradas (picos emocionales claros).
- Agrupamos emociones similares (por ejemplo, positivo vs. negativo).

Edad (fuerte en 'anciano' pero borroso en adulto/joven/niño)

Las Edades han dado muchos conflictos, ya que para encontrar imágenes ha sido la categoría más complicada, ya que una persona de unos 30 años tiende a verse más adulta que una persona

de 18 años, pero no llega a ser tan adulto como una de 45 años, así que eso genera confusión para nosotros al no saber catalogar franjas de edad claras.

Mejoras posibles:

- Aumentar los datos en franjas de edad conflictivas (joven-adulto).
- Reagruparíamos los rangos con límites más homogéneos (por ejemplo, ‘0–12’, ‘13–35’, ‘36–60’, ‘60+’).

6. Integración del modelo LLM y ejemplos generados

6.1 Cómo se construyen los prompts a partir de los resultados.

Primero, preprocesamos la imagen enviada por el usuario y se la enviamos a cada uno de los modelos. Una vez obtenidas las predicciones y sus respectivos porcentajes de probabilidad, recorremos los resultados y para cada categoría evaluamos si la predicción más alta supera el 60%.

Si supera ese umbral, simplemente añadimos al prompt una frase del tipo: "Hombre al 85%", usando directamente la etiqueta más probable.

```
prompt = f"""
Eres un narrador creativo con mucho sentido del humor.

Tu tarea es escribir una historia divertida y breve (máximo 45 palabras) basada en características predecidas por una IA de visión artificial. Las predicciones incluyen atributos como edad, género, emociones, profesión, entre otros, junto con su porcentaje de certeza.

Si una etiqueta supera el 60% de probabilidad, úsala directamente en la historia, que se mencione claramente, solo en el caso de que supere el 60%.
Si ninguna etiqueta supera el 60% en una categoría, usa una descripción ambigua o mezcla de posibilidades
No hables nunca de dos personas cuando haya lo comentado antes de etiquetas ambiguas.
Siempre centrate en una sola y le haces la descripción amiga.
(por ejemplo, "alguien que podría ser camarero o médico").
La historia debe ser coherente, creativa, graciosa y visualmente sugerente.
No menciones porcentajes ni el hecho de que son predicciones.
No expliques lo que estás haciendo, solo escribe la historia.

Etiquetas con porcentajes:
{chr(10).join(partes)}
"""

"""

```

Imagen 40

Si ninguna categoría supera el 60%, optamos por una descripción más ambigua, incluyendo las tres etiquetas con mayor probabilidad, por ejemplo: "entre Médico 40% / Camarero 35% / Policía 25%".

Estas frases las concatenamos en un único bloque que usamos como parte del contenido del prompt. Luego, le indicamos al modelo que genere una historia divertida de aproximadamente 40 palabras basada en esa descripción.

6.2 Qué modelo de lenguaje se ha utilizado (API externa o modelo local).

Cuando subimos una imagen desde el frontend (por ejemplo, una cara o una foto cualquiera), el servidor recibe ese archivo y lo procesa. Lo primero que hacemos es comprobar que sea una imagen válida (tipo .jpg, .png, etc.), y luego leemos los bytes una sola vez para no sobrecargar, se preprocesa y se envía al LLM.

En esta API **hemos utilizado un modelo de lenguaje a través de una API externa de OpenAI**, específicamente el modelo **gpt-3.5-turbo**. Esto lo vemos claramente en la parte del código donde usamos el cliente de OpenAI para generar una historia creativa basada en las predicciones de los modelos, estamos enviando una solicitud a los servidores de OpenAI, lo que significa que **no estamos usando un modelo local**, sino uno externamente.

```
app = Flask(__name__, template_folder='templates', static_folder='static')
cors(app)

# Inicializar OpenAI
client = openai.Client(api_key=os.getenv("OPENAI_API_KEY", "sk-proj-mFmSz2hoFN30RVsy7unyL1l1HmzEwzcRwvqs-i" \
"dlKFu9jNA7YgkG4g7s2fRvzE6941-ShWFnTqT3R1bkFJeoktsVqftRzy7Warak9sL6RhFQFpR0LE6t91x9rjf1Lcu7a8eGawRNcDA_PSiNtpH1tC-bnmCQA"))

CLASES_MODELOS = {
    'sexo': ['Hombre', 'Mujer'],
    'gafas': ['Con Gafas', 'Sin Gafas'],
    'profesiones': ['Bombero', 'Camarero', 'Medico', 'Policia'],
    'emociones': ['Enfadado', 'Feliz', 'Neutral', 'Triste'],
    'pelo': ['Pelo Corto', 'Pelo Largo'],
    'edad': ['Adulto', 'Anciano', 'Joven', 'Niño']
}
```

Imagen 41

6.3 Ejemplos de textos generados.

A continuación, vamos a ver algunos de los textos que hemos conseguido con nuestro prompt a Chat GPT 3.5 Turbo.



Imagen 42



Imagen 43

6.4 Valoración de los resultados y posibles mejoras.

En nuestra valoración, las microhistorias suelen reflejar bien las etiquetas con alta confianza, pero a veces la ambigüedad se alarga demasiado y rompe el ritmo. Para pulir esto, podríamos establecer un límite máximo de dos opciones ambiguas por categoría y usar conectores más fluidos (“quizá”, “tal vez”) que suavicen la transición entre ideas.

Además, hemos visto que el lenguaje tiende a caer en patrones repetitivos, como empezar siempre con “un día” o “de repente”. Para darle más chispa, podríamos preparar un set de miniplantillas con distintas aperturas y cierres, eligiendo una al azar para cada historia. Así mantenemos la narración fresca sin sobrecargar el prompt.

También, sería interesante probar con otro LLM —por ejemplo, un modelo de Microsoft o Google— para comparar resultados. Podríamos ejecutar la misma lógica de generación en paralelo, medir métricas básicas (longitud, coherencia, variedad léxica) y decidir dinámicamente cuál ofrece la mejor mezcla de creatividad y solidez. De ese modo, aprovechamos fortalezas de distintos modelos y tenemos siempre la opción mejor ajustada.

Finalmente, sería útil poner en marcha un bucle de feedback automático: generar varias versiones de cada microhistoria, evaluarlas con métricas sencillas (longitud, variedad léxica, coherencia) y quedarnos con la mejor. Incluso podríamos traducir estas métricas en puntuaciones y ajustar el prompt o los umbrales de ambigüedad en tiempo real, mejorando la calidad con cada iteración.

7. Descripción de la aplicación web

7.1 Cómo está estructurada la aplicación (frontend/backend).

La aplicación está estructurada en dos partes principales: el frontend (interfaz de usuario) y el backend (lógica del servidor y modelo de clasificación).

- Frontend:

La interfaz de usuario está desarrollada como una página web que permite al usuario subir una imagen desde su dispositivo. Esta interfaz es sencilla e intuitiva, pensada para facilitar la interacción con el sistema. Utiliza tecnologías estándar como HTML, CSS y JavaScript para mostrar formularios, botones y resultados de manera clara. Una vez que el usuario selecciona una imagen, esta se envía al backend mediante una petición HTTP, generalmente a través de fetch o axios en JavaScript.

- Backend (API Flask en Python):

El backend está desarrollado en Python utilizando el framework Flask, que permite crear una API REST para recibir las imágenes enviadas por el usuario. Esta API se encarga de:

1. Recibir y procesar la imagen (cambio de tamaño, normalización, etc.).
2. Pasar la imagen al modelo de redes neuronales convolucionales (CNN) previamente entrenado, que realiza la predicción de múltiples atributos (profesión, edad, sexo, emoción y uso de gafas).
3. Devolver la predicción al frontend, en forma de JSON, con las etiquetas clasificadas para esa imagen.

Esta arquitectura cliente-servidor permite separar claramente las responsabilidades de cada parte: el frontend se enfoca en la experiencia del usuario y el backend en el procesamiento de datos y la

inteligencia artificial. Además, esta estructura facilita la escalabilidad del sistema, ya que permite sustituir o mejorar cada componente de forma independiente.

7.2 Flujo de funcionamiento desde la subida de imagen hasta la generación del texto.

El flujo de funcionamiento de la aplicación sigue una serie de pasos organizados entre el **frontend** y el **backend**, desde que el usuario sube una imagen hasta que recibe el resultado con la clasificación. A continuación, se describe este proceso paso a paso:

1. Subida de imagen (Frontend):

El usuario accede a la página web y selecciona una imagen desde su dispositivo. Esta imagen representa el rostro o cuerpo de una persona cuya información se quiere clasificar (profesión, edad, sexo, emoción, gafas).

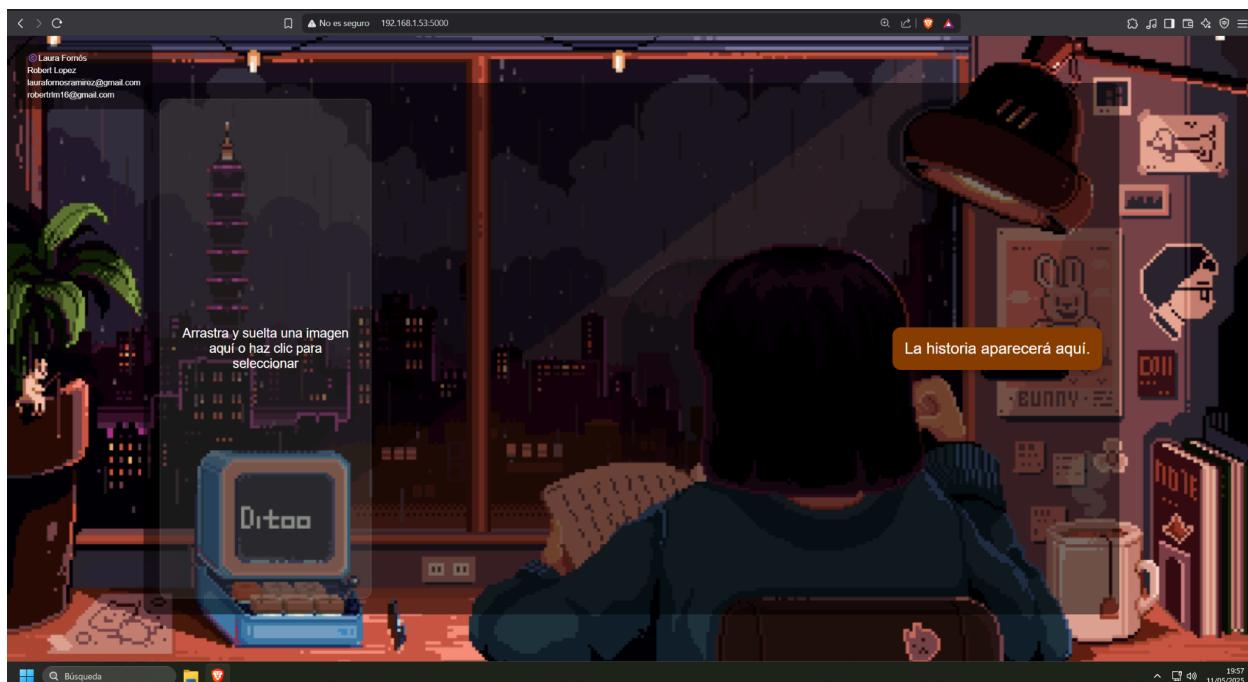


Imagen 44

2. Envío al servidor (Frontend → Backend):

Una vez seleccionada, la imagen se envía al servidor mediante una petición HTTP (POST) a través de la API Flask. Esta petición incluye la imagen en formato adecuado.

3. Recepción y preprocesamiento de la imagen (Backend):

El servidor Flask recibe la imagen y realiza tareas de preprocesamiento como el cambio de tamaño, normalización de píxeles y transformación a un formato que pueda ser interpretado por el modelo de Tensor.

4. Clasificación con el modelo CNN (Backend):

La imagen preprocesada se introduce en el modelo de redes neuronales convolucionales (CNN), que ha sido previamente entrenado con imágenes etiquetadas. El modelo genera una predicción para cada una de las categorías definidas:

5. Generación del resultado en formato texto (Backend):

A partir de las predicciones del modelo, el backend construye un texto explicativo a través de una llamada con un prompt a Chat Gpt.

6. Respuesta al cliente (Backend → Frontend):

Este texto se envía como respuesta en formato JSON al navegador del usuario, junto con las etiquetas si es necesario.

7. Visualización del resultado (Frontend):

El frontend recibe el resultado y lo muestra en pantalla, permitiendo al usuario ver la descripción generada por la inteligencia artificial a partir de su imagen.



Imagen 45

8. Conclusiones y reflexión final

8.1 Qué se ha aprendido en el desarrollo del proyecto.

A lo largo del desarrollo de este proyecto, hemos adquirido un conocimiento profundo y práctico sobre el funcionamiento de redes neuronales convolucionales (CNN) aplicadas a la clasificación de imágenes. Hemos aprendido a preparar un dataset adecuado, incluyendo la recolección y limpieza de imágenes provenientes de distintos bancos, así como su etiquetado correcto en múltiples categorías (profesión, edad, emoción, sexo y uso de gafas). También se ha profundizado en el diseño e implementación de una API RESTful con Flask para conectar el modelo de inteligencia artificial con una interfaz web funcional. En este proceso, se han reforzado habilidades de programación en Python, manejo de frameworks de desarrollo web y conceptos clave de inteligencia artificial y big data.

8.2 Qué partes han sido más difíciles o interesantes.

Una de las partes más desafiantes ha sido el entrenamiento del modelo CNN con múltiples etiquetas simultáneamente, ya que requiere un diseño cuidadoso tanto en la arquitectura de la red como en el manejo de la salida para que pueda predecir correctamente atributos diversos de una misma imagen. También ha resultado compleja la gestión del dataset, ya que ha sido necesario equilibrar las clases para evitar sesgos en el modelo y asegurar una buena generalización. Por otro lado, la parte más interesante ha sido comprobar cómo el modelo es capaz de reconocer correctamente características visuales sutiles, como una emoción o si una persona lleva gafas, lo que demuestra el potencial de la inteligencia artificial en tareas de análisis de imagen.

8.3 Qué harías diferente si volvierais a empezar

Si tuviéramos que empezar de nuevo el proyecto, planificaríamos de forma más meticulosa la estructura y balance del dataset desde el principio, ya que la calidad y diversidad de los datos ha sido un factor clave para el rendimiento del modelo. También estudiaríamos el uso de modelos preentrenados mediante transferencia de aprendizaje (como MobileNet o ResNet), lo que podría haber reducido los tiempos de entrenamiento y mejorado la precisión con menos datos. Además, dedicaríamos más tiempo al diseño de la interfaz web y la experiencia del usuario, para que la aplicación resultara más accesible e intuitiva. Finalmente, implementaríamos métricas más completas para evaluar el rendimiento del modelo en cada una de las categorías de clasificación.

9. Bibliografía

9.1 Lista de Imágenes

1. [Imagen 1](#): Trello
2. [Imagen 2](#): Imagenes Robert
3. [Imagen 3](#): Imagenes Laura
4. [Imagen 4](#): Carpeta Programa Backup
5. [Imagen 5](#): Códigos Entrenamientos
6. [Imagen 6](#): Carpeta Static
7. [Imagen 7](#): Carpeta Programa
8. [Imagen 8](#): Carpeta Drive Credit de Síntesis
9. [Imagen 9](#): Carpeta Documentos
10. [Imagen 10](#): Definimos el nombre de la carpeta y donde la guardamos.
11. [Imagen 11](#): Definimos el nombre de las imágenes que descargamos.
12. [Imagen 12](#): Reporte Clasificación Sexo.
13. [Imagen 13](#): Reporte Clasificación Gafas
14. [Imagen 14](#): Reporte Clasificación Profesiones.
15. [Imagen 15](#): Reporte Clasificación Edad.
16. [Imagen 16](#): Reporte Clasificación Emociones.
17. [Imagen 17](#): Reporte Clasificación Pelo.
18. [Imagen 18](#): Matriz de Confusión Sexo.
19. [Imagen 19](#): Matriz de Confusión Gafas.
20. [Imagen 20](#): Matriz de Confusión Profesiones.
21. [Imagen 21](#): Matriz de Confusión Edad.
22. [Imagen 22](#): Matriz de Confusión Emociones.
23. [Imagen 23](#): Matriz de Confusión Pelo.
24. [Imagen 24](#): Antes reporte de clasificación Sexo
25. [Imagen 25](#): Antes matriz de confusión Sexo
26. [Imagen 26](#): Antes reporte de clasificación Gafas
27. [Imagen 27](#): Antes matriz de confusión Gafas
28. [Imagen 28](#): Antes reporte de clasificación Profesiones

29. [Imagen 29](#): Antes matriz de confusión Profesiones.
30. [Imagen 30](#): Imagen de Camarero preprocesada.
31. [Imagen 31](#): Imagen de Médico preprocesada.
32. [Imagen 32](#): Antes reporte de clasificación Edad
33. [Imagen 33](#): Antes matriz de confusión Edad
34. [Imagen 34](#): Antes reporte de clasificación Emociones
35. [Imagen 35](#): Antes matriz de confusión Emociones
36. [Imagen 36](#): Antes reporte de clasificación Pelo
37. [Imagen 37](#): Antes matriz de confusión Pelo
38. [Imagen 38](#): Imagen Pelo Corto preprocesada,
39. [Imagen 39](#): Imagen Pelo Largo preprocesada.
40. [Imagen 40](#): Prompt en la API para ChatGPT 3.5 Turbo
41. [Imagen 41](#): Inicialización de la OpenAI Key y la API
42. [Imagen 42](#): Prueba 1 del Prompt
43. [Imagen 43](#): Prueba 2 del Prompt
44. [Imagen 44](#): Interfaz Web
45. [Imagen 45](#): Funcionamiento Modelos Web

9.2 Lista de Links de Paginas Web utilizadas para el Web Scraping

1. <https://scrapingant.com/blog/how-to-scrape-google-images>
2. <https://www.kaggle.com>
3. <https://es.pinterest.com>
4. https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.htm
5. <https://github.com/danielgatis/rembg>
6. <https://www.shutterstock.com>
7. https://www.vecteezy.com/search?content_type=photo&qterm=policeman-portrait
8. <https://www.freepik.es>
9. <https://flypix.ai/es/blog/image-recognition-in-python>
10. <https://www.kaggle.com/datasets/pooriamst/age-prediction>
11. <https://www.kaggle.com/datasets/abdullah0a/human-age-prediction-synthetic-dataset>
12. <https://www.ibm.com/es-es/think/topics/convolutional-neural-networks>
13. <https://www.isdi.education/es/blog/redes-neuronales-convolucionales>

9.3 Instalaciones en VS Code

1. pip install tensorflow
2. pip install opencv-python
3. pip install scikit-learn
4. pip install pillow
5. pip install rembg
6. pip install flask
7. pip install flask-cors
8. pip install openai
9. pip install selenium
10. pip install webdriver-manager
11. pip install tqdm
12. pip install seaborn
13. pip install matplotlib