

**Universidad Nacional Siglo XX**  
**ESAM Escuela de Negocios**  
**MAESTRÍA EN BIG DATA Y ANALÍTICA DATOS**  
**MÓDULO 11 – REDES NEURONALES Y DEEP LEARNING**



**Trabajo Final**

**Reconocimiento Facial con Redes Neuronales: Un Caso Práctico con Celebridades y Aplicación en Empresas**

**MAESTRANTES:**

Paul Caihuara Caba

Gavino Abdel Gonzales Calle

Marcelo Fernando Condori Mendoza

02 de agosto de 2024

## **1. Introducción:**

En este trabajo, se propone desarrollar un sistema de reconocimiento facial con un objetivo más específico: identificar a los empleados de una empresa o institución. Se piensa en un escenario donde al ingresar a una oficina, las puertas se desbloquean automáticamente al reconocer el rostro del empleado. Otra alternativa de implementación puede basarse en un sistema de control de acceso que registre la hora de **entrada y salida** de cada persona.

Para lograr este objetivo, se han empleado redes neuronales convolucionales (CNN), una arquitectura de aprendizaje profundo especialmente diseñada para el procesamiento de imágenes. Estas redes han demostrado ser altamente efectivas en tareas de clasificación de imágenes, como el reconocimiento facial.

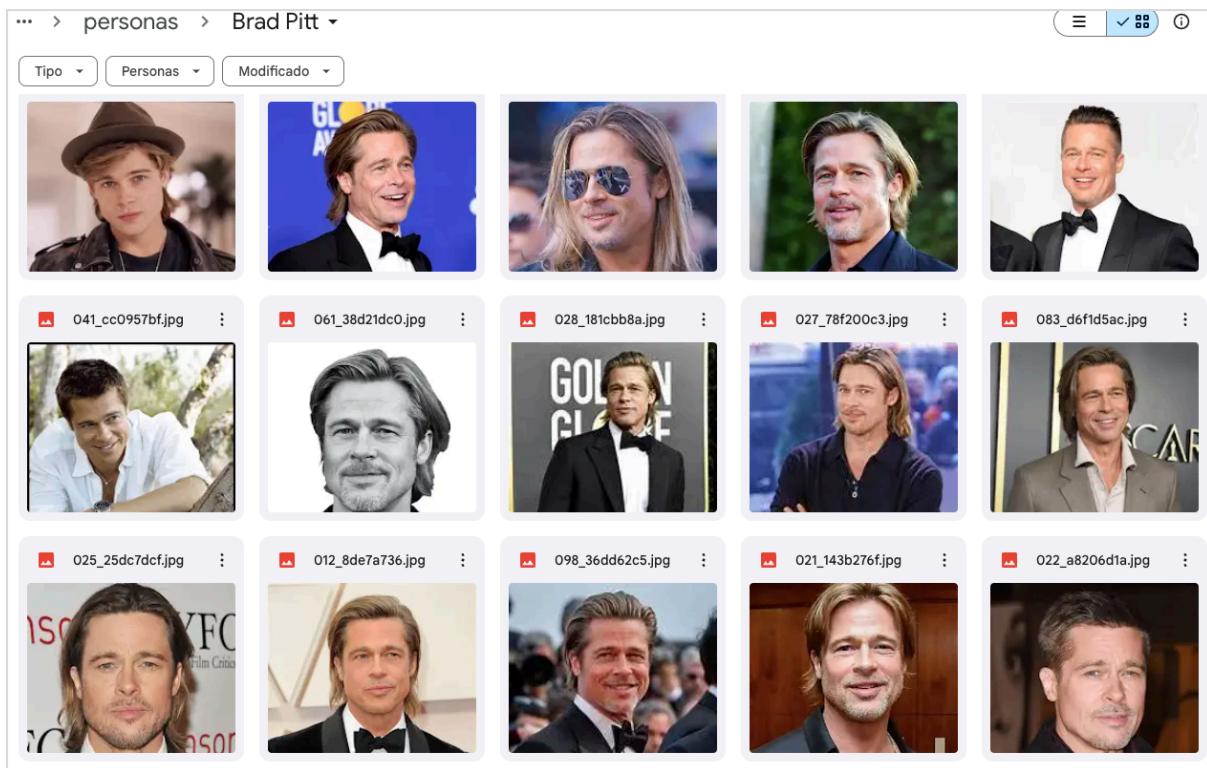
## **2. Obtención y Preprocesamiento de Datos:**

Para entrenar el modelo de reconocimiento facial, se inició la recopilación de un conjunto de datos de imágenes de rostros de celebridades. Este conjunto fue obtenido de la plataforma Kaggle. Sin embargo, los datos en bruto requerían un procesamiento previo para adecuarlos a las necesidades del modelo implementado.

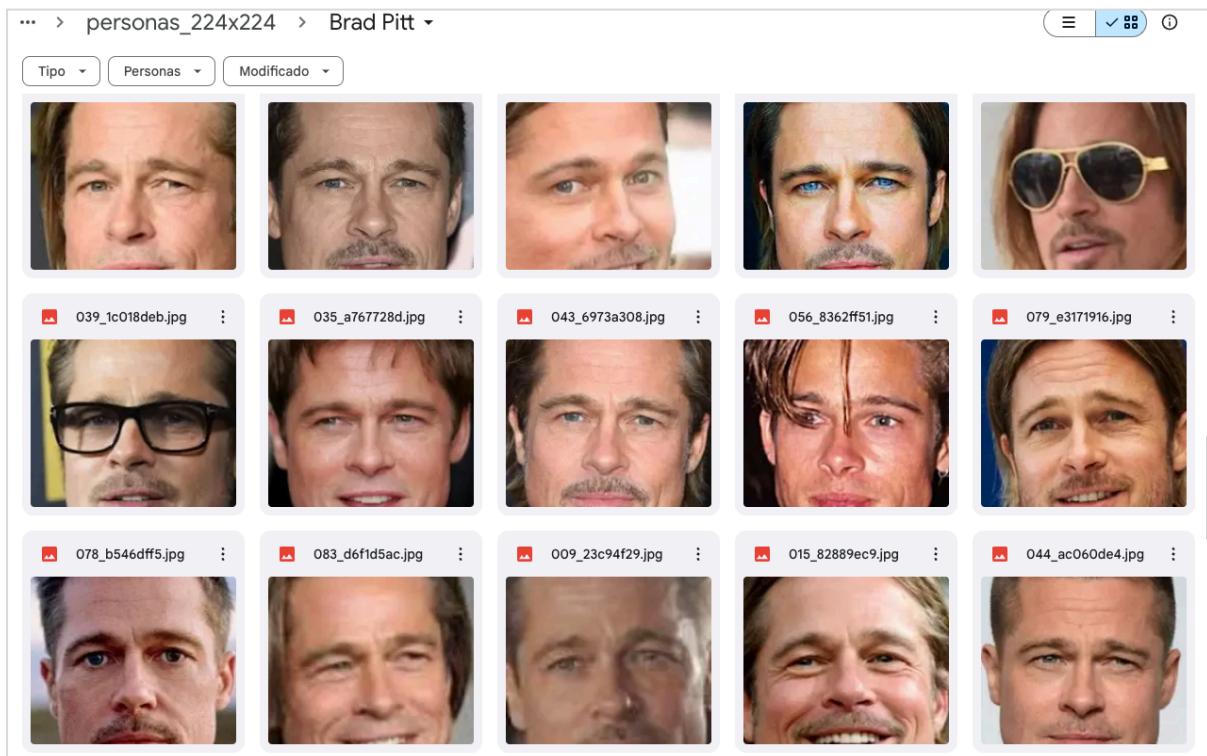
The screenshot shows a dark-themed Kaggle dataset page. At the top, the title "Celebrity Face Image Dataset" is displayed in large white font. Below it, a subtitle reads "1800+ images of celebrity faces of 18 people!". A navigation bar at the top right includes links for "Data Card", "Code (51)", "Discussion (0)", and "Suggestions (0)". A horizontal line separates this from the main content. The "About Dataset" section is titled "About Dataset" in bold white font. Below it, a descriptive text states: "This dataset contains images of 18 Hollywood celebrities with 100 images of each celebrity. The people in this dataset are:". A bulleted list follows, naming the 18 celebrities: Angelina Jolie, Brad Pitt, Denzel Washington, Hugh Jackman, Jennifer Lawrence, and Johnny Depp.

<https://www.kaggle.com/datasets/vishesh1412/celebrity-face-image-dataset>

Utilizando la biblioteca OpenCV, se desarrollaron una serie de scripts para detectar los rostros en cada imagen, recortarlos y redimensionarlos a un tamaño específico. Este proceso fue esencial para garantizar que nuestra red neuronal recibiera imágenes con características similares, y optimizar así el aprendizaje.



Imágenes antes de procesar (algunas de cuerpo completo)



Imágenes procesadas (solamente sus rostros)

Recopilación y preprocesamiento de datos: Se utilizó un conjunto de datos de imágenes de celebridades como punto de partida, adaptándolo a nuestras necesidades mediante técnicas de detección y recorte de rostros.

### 3. Arquitectura de la Red Neuronal y Entrenamiento

Para abordar el complejo problema del reconocimiento facial, seleccionamos una arquitectura de red neuronal convolucional (CNN). Estas redes están especialmente diseñadas para procesar imágenes, extrayendo características relevantes como bordes, texturas y formas. Nuestra CNN constaba de múltiples capas convolucionales, seguidas de capas de *pooling* para reducir la dimensionalidad y capas densamente conectadas para la clasificación final.

El entrenamiento de la red se llevó a cabo utilizando un gran conjunto de datos de imágenes de rostros. A través de un proceso iterativo, la red ajustaba sus parámetros internos para minimizar la diferencia entre las predicciones y las etiquetas verdaderas. Utilizamos la función de pérdida de entropía cruzada categorizada y el optimizador Adam para este proceso.

```
# Definir el modelo
def create_model(num_classes):
    model = Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model
```

Parte de fragmento de notebook de entrenamiento

**Diseño y entrenamiento de la red neuronal:** Se construyó una CNN personalizada y se aplicó el entrenamiento utilizando un gran número de imágenes de rostros.

### 4. Evaluación del Modelo:

Una vez entrenada la red neuronal, se procedió a evaluar su desempeño en un conjunto de datos de prueba que no había sido utilizado durante el entrenamiento. Para ello, se emplearon métricas de evaluación estándar como la precisión, el *recall* y la *F1-score*. Estas métricas nos permitieron cuantificar la capacidad del modelo para clasificar correctamente los rostros y distinguir entre diferentes individuos.

Los resultados obtenidos indicaron que nuestro modelo alcanzaba una precisión satisfactoria en la tarea de reconocimiento facial, demostrando su capacidad para generalizar a nuevos datos. Sin embargo, identificamos algunas limitaciones, como la dificultad para distinguir entre individuos con características faciales muy similares.

```
33/33 10s 224ms/step - accuracy: 0.9825 - loss: 0.0794 - val_accuracy: 0.6908 - val_loss: 1.7676
Epoch 17/20
33/33 9s 179ms/step - accuracy: 0.9806 - loss: 0.0757 - val_accuracy: 0.6985 - val_loss: 1.8376
Epoch 18/20
33/33 8s 247ms/step - accuracy: 0.9631 - loss: 0.1046 - val_accuracy: 0.7099 - val_loss: 1.9608
Epoch 19/20
33/33 7s 210ms/step - accuracy: 0.9951 - loss: 0.0222 - val_accuracy: 0.7099 - val_loss: 1.8739
Epoch 20/20
33/33 11s 217ms/step - accuracy: 0.9991 - loss: 0.0123 - val_accuracy: 0.7061 - val_loss: 1.9081

[14] # Evaluar el modelo
    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
    print(f'\nPrecisión en el conjunto de prueba: {test_acc}')

→ 11/11 - 0s - 43ms/step - accuracy: 0.6361 - loss: 2.4441
Precisión en el conjunto de prueba: 0.6360856294631958
```

*Parte de fragmento de notebook de entrenamiento*

Verificación de funcionamiento del modelo, pasándole una imagen de internet el modelo responde adecuadamente.

```
+ Código + Texto
[15] # MOSTRAR EL RESULTADO
if identified_person:
    plt.title(f'Persona identificada: {identified_person} \n Confianza: {confidence:.2f}')
    print(f'Persona identificada: {identified_person} \n con confianza: {confidence:.2f}')
else:
    plt.title(f'No se pudo identificar a la persona \n Confianza: {confidence:.2f}')
    print(f'No se pudo identificar a la persona \n Confianza: {confidence:.2f}')


img = mpimg.imread(new_image_path)
imgplot = plt.imshow(img)
plt.show()

→ 1/1 [=====] - 0s 225ms/step
Persona identificada: Megan Fox
con confianza: 1.00
    Persona identificada: Megan Fox
    Confianza: 1.00
    
```

*Verificación con una imagen de internet*

## 5. Implementación como Microservicio

Una vez que tuvimos un modelo de reconocimiento facial entrenado y evaluado, el siguiente paso fue desplegarlo como un microservicio para que pudiera ser utilizado en aplicaciones reales. Para ello, elegimos la popular biblioteca *Python FastAPI* debido a su facilidad de uso y alto rendimiento.

Utilizando Google Colab como entorno de desarrollo, creamos un notebook de Python donde cargamos el modelo entrenado y las etiquetas correspondientes. A continuación, definimos los endpoints de nuestra API utilizando FastAPI. Estos endpoints exponen las funcionalidades del modelo, permitiendo a los clientes enviar imágenes y obtener las predicciones correspondientes.



*Exponiendo microservicio al mundo (con fines de testing)*

The screenshot shows a Swagger UI interface with the following details:

- Request URL:** `https://cb57-34-106-47-40.ngrok-free.app/identify`
- Server response:**

Code	Details
200	<b>Response body</b> <pre>{   "person": "Megan Fox",   "confidence": 0.99895179271698 }</pre> <span style="float: right;">Copy Download</span>

- Response headers**  

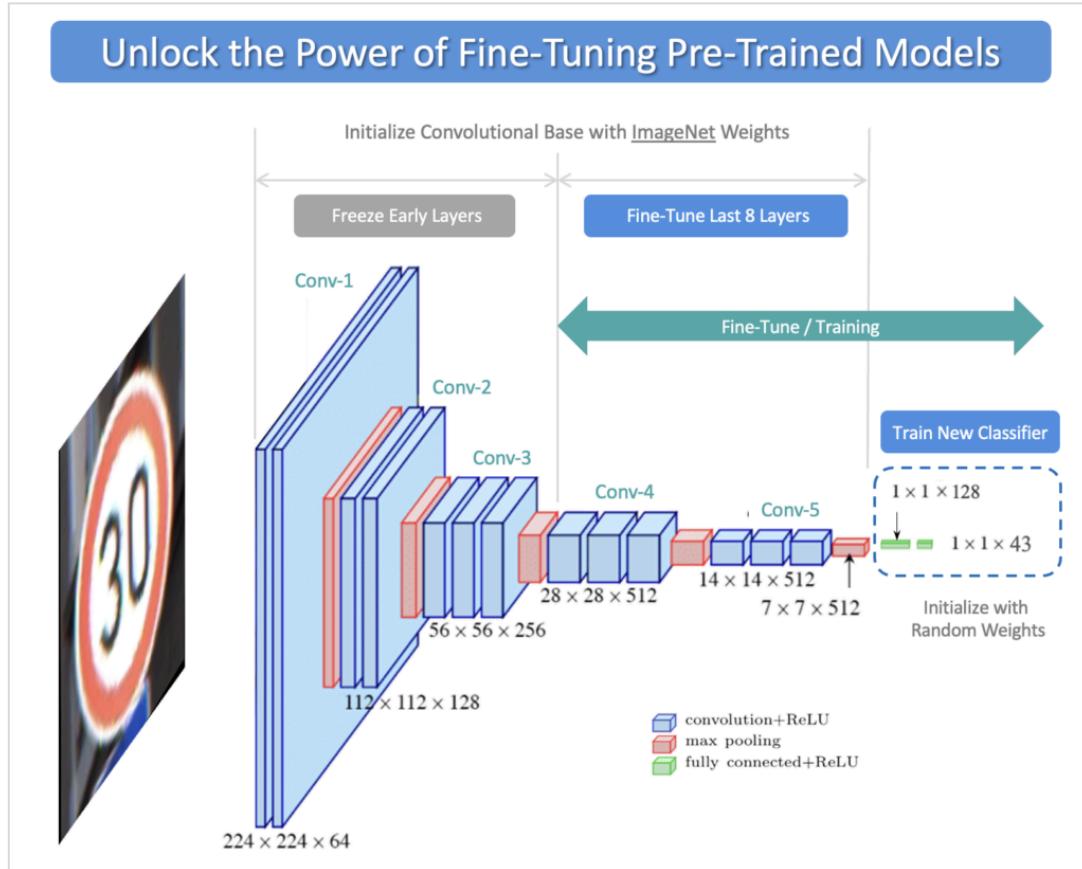
```
access-control-allow-credentials: true  
access-control-allow-origin: https://cb57-34-106-47-40.ngrok-free.app  
content-length: 52  
content-type: application/json  
date: Tue, 30 Jul 2024 20:48:11 GMT  
server: uvicorn  
vary: Origin
```

*Swagger del microservicio*

Para hacer nuestro microservicio accesible desde cualquier parte del mundo, utilizamos la herramienta Ngrok. Esta herramienta crea un túnel seguro que redirige las solicitudes HTTP a nuestro notebook en Google Colab. De esta manera, pudimos obtener una URL pública a la que se podían enviar solicitudes a nuestra API.

## 6. Fine-tuning y Mantenimiento del Modelo

El reconocimiento facial es un campo en constante evolución, y los modelos deben adaptarse a nuevos datos y escenarios. El fine-tuning es una técnica que permite ajustar un modelo pre-entrenado a un conjunto de datos específico, en este caso como un nuevo conjunto de empleados que ingresen a la empresa, por lo tanto es necesario actualizar el modelo entrenado con nuevos datos.



Concepto de Fine Tuning

Sin embargo, es crucial tener en cuenta que el fine-tuning puede llevar a un fenómeno conocido como catástrofe del olvido, donde el modelo puede olvidar información aprendida previamente. Para mitigar este problema, se pueden emplear técnicas como la regularización y el aprendizaje incremental.

```

Epoch 17/20
42/42 [=====] - 3s 72ms/step - loss: 1.5564 - accuracy: 0.5407 - val_loss: 7.7562 - val_accuracy: 0.0601
Epoch 18/20
42/42 [=====] - 4s 92ms/step - loss: 1.4980 - accuracy: 0.5542 - val_loss: 7.9712 - val_accuracy: 0.0541
Epoch 19/20
42/42 [=====] - 4s 107ms/step - loss: 1.4447 - accuracy: 0.5663 - val_loss: 8.1096 - val_accuracy: 0.0601
Epoch 20/20
42/42 [=====] - 3s 70ms/step - loss: 1.3893 - accuracy: 0.5904 - val_loss: 8.2001 - val_accuracy: 0.0631

[ ] def save_labels(labels, filename):
    labels = {k: int(v) for k, v in labels.items()}
    with open(filename, 'w') as f:
        json.dump(labels, f)

[ ] # Guardar el modelo y las etiquetas actualizadas
model.save('/content/drive/MyDrive/Personales/Estudio/BigData/Mod11/TareaFinal/face_recognition_model_updated.h5')

def save_labels(labels, filename):
    labels = {k: int(v) for k, v in labels.items()}
    with open(filename, 'w') as f:
        json.dump(labels, f)

save_labels(label_mapping, '/content/drive/MyDrive/Personales/Estudio/BigData/Mod11/TareaFinal/face_recognition_model_updated.json')
print("Modelo y etiquetas actualizados guardados exitosamente.")

→ Modelo y etiquetas actualizados guardados exitosamente.

```

*Fragmento del notebook donde se actualiza el modelo y los tags*

## 7. Adjuntos

Los siguiente notebooks, realizados en Google Colab.

- M11\_Final\_01RecFacial-**procesar.ipynb**
- M11\_Final\_02RecFacial-**entrenar.ipynb**
- M11\_Final\_03RecFacial-**verificar.ipynb**
- M11\_Final\_04RecFacial-**fine-tunning.ipynb**
- M11\_Final\_05RecFacial-**microservicio.ipynb**

### 7.1. Procesamiento Inicial

A continuación se aprecia la ejecución del procesamiento inicial “M11\_Final\_01RecFacial-**procesar.ipynb**”.

```

✓ [1] from google.colab import drive
# Montar Google Drive
drive.mount('/content/drive')

→ Mounted at /content/drive

✓ [2] pip install opencv-python tqdm
→ Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (4.66.4)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python) (1.26.4)

✓ [3] #Vamos a preprocesar imágenes cortando solamente sus rostros
import os
import cv2
import numpy as np
from tqdm import tqdm

def preprocess_image(image, target_size=(224, 224)):
    # Convertir a RGB si es necesario
    if len(image.shape) == 2 or image.shape[2] == 1:
        image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
    elif image.shape[2] == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Detección facial
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    if len(faces) > 0:
        x, y, w, h = faces[0]
        face = image[y:y+h, x:x+w]
    else:
        print("No se detectó ninguna cara, usando la imagen completa.")
        #face = image
        return None

    # Redimensionar
    face = cv2.resize(face, target_size)

```

Se aplica también el procesamiento de imágenes y la posterior descompresión de los archivos.

```
[3] # Redimensionar
face = cv2.resize(face, target_size)

return face

def process_directory(input_dir, output_dir):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for person_folder in tqdm(os.listdir(input_dir)):
        person_path = os.path.join(input_dir, person_folder)
        if os.path.isdir(person_path):
            output_person_path = os.path.join(output_dir, person_folder)
            if not os.path.exists(output_person_path):
                os.makedirs(output_person_path)

            for image_file in os.listdir(person_path):
                if image_file.lower().endswith('.png', '.jpg', '.jpeg'):
                    input_image_path = os.path.join(person_path, image_file)
                    output_image_path = os.path.join(output_person_path, image_file)

                    image = cv2.imread(input_image_path)
                    processed_image = preprocess_image(image)
                    if processed_image is not None:
                        cv2.imwrite(output_image_path, cv2.cvtColor(processed_image, cv2.COLOR_RGB2BGR))

[ ] # Descompresión de archivos
!unzip /content/archivos_imagenes.zip
→ inflating: Celebrity Faces Dataset/Scarlett Johansson/090_8e8d0b5c.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/091_764fb4f6.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/092_4c27282f.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/093_fdaef117.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/094_ad3f1bcb.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/095_8c6b7820.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/096_9d0cd927.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/097_ca9cfda8.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/098_c71a52d0.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/099_1046ebbd.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/100_0bc6635b.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/101_7850b100.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/102_97fcfa716.jpg
inflating: Celebrity Faces Dataset/Scarlett Johansson/103_c4026864.jpg
```

```

Mounted at /content/drive

# Cargar datos
# dir_path = '/content/drive/MyDrive/Personales/Estudio/BigData/Mod11/TareaFinal/personas_224x224'
dir_path = '/content/drive/MyDrive/ProyectoFotos/Celebrity_Faces_Dataset/personas_nuevas_224x224/'
images, labels = load_data(dir_path)

...

```

Seguidamente se aplica el entrenamiento del modelo a través de la ejecución del Notebook “M11\_Final\_02RecFacial-entrenar.ipynb”.

```

M11_Final_02RecFacial-entrenar.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se guardaron todos los cambios
Comentar Compartir Géminis RAM Disco
+ Código + Texto

Q
▼ 02 Entrenar modelo
{x}
Con el directorio especificado se procede a entrenar

[1] import numpy as np
import os
import cv2
import tensorflow
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

[2] import tensorflow as tf
from tensorflow.keras import layers

[3] # Configuración
IMG_SIZE = 64
BATCH_SIZE = 32
EPOCHS = 20

[4] # Función para cargar imágenes y etiquetas
def load_data(dir_path):
    images = []
    labels = []
    for person_name in os.listdir(dir_path):
        person_path = os.path.join(dir_path, person_name)
        if os.path.isdir(person_path):
            for img_name in os.listdir(person_path):
                img_path = os.path.join(person_path, img_name)
                img = cv2.imread(img_path)
                img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
                img = img / 255.0 # BGR to RGB and normalize
                images.append(img)
                labels.append(person_name)
array(labels)

```

Activate Windows  
Go to Settings to activate Windows.

Se guardó correctamente 0 s se ejecutó 11:45 a.m.

Además, se define el modelo a través de la red neuronal convolucional compuesta por ocho capas.

```
+ Código + Texto
✓ 0s [4]         img = img[:, :, ::-1] / 255.0 # BGR to RGB and normalize
✓ 0s [4]         images.append(img)
✓ 0s [4]         labels.append(person_name)
✓ 0s [4]         return np.array(images), np.array(labels)

✓ 0s [5] # Definir el modelo
def create_model(num_classes):
    model = Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model

✓ 0s [6] def save_labels(labels, filename):
    # Convert NumPy integers to Python integers
    labels = {k: int(v) for k, v in labels.items()}

    with open(filename, 'w') as f:
        json.dump(labels, f)

✓ 20s [7] from google.colab import drive
# Montar Google Drive
drive.mount('/content/drive')

→ Mounted at /content/drive

✓ 4m [9] # Cargar datos
# dir_path = '/content/drive/MyDrive/Personales/Estudio/BigData/Mod11/TareaFinal/personas_224x224'
dir_path = '/content/drive/MyDrive/ProyectoFotos/Celebrity_Faces_Dataset/personas_nuevas_224x224/'
images, labels = load_data(dir_path)
```

```
✓ 0s [10] # Codificar etiquetas
le = LabelEncoder()
labels_encoded = le.fit_transform(labels)

# Guardar el mapeo de etiquetas
label_mapping = dict(zip(le.classes_, le.transform(le.classes_)))

✓ 0s [11] label_mapping
→ {'Angelina Jolie': 0,
 'Brad Pitt': 1,
 'Denzel Washington': 2,
 'Hugh Jackman': 3,
 'Jennifer Lawrence': 4,
 'Johnny Depp': 5,
 'Kate Winslet': 6,
 'Leonardo DiCaprio': 7,
 'Megan Fox': 8,
 'Natalie Portman': 9,
 'Nicole Kidman': 10,
 'Robert Downey Jr': 11,
 'Sandra Bullock': 12,
 'Scarlett Johansson': 13,
 'Tom Cruise': 14,
 'Tom Hanks': 15,
 'Will Smith': 16}

✓ 0s [12] # save_labels(label_mapping, '/content/drive/MyDrive/Personales/Estudio/BigData/Mod11/TareaFinal/face_recognition_model.json')
save_labels(label_mapping, '/content/drive/MyDrive/ProyectoFotos/JSON/face_recognition_model.json')

# Convertir a one-hot encoding
labels_one_hot = to_categorical(labels_encoded)

# Dividir datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(images, labels_one_hot, test_size=0.2, random_state=42)
```

```

[13] # Crear y compilar el modelo
num_classes = len(np.unique(labels_encoded))
model = create_model(num_classes)
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Entrenar el modelo
history = model.fit(X_train, y_train, epochs=EPOCHS, validation_split=0.2, batch_size=BATCH_SIZE)

1s [13] # /usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/'input_dim` argument to a layer. When using Sequential models, prefer super().__init__(activity_regularizer, **kwargs)
Epoch 1/20
33/33    12s 216ms/step - accuracy: 0.0800 - loss: 2.8368 - val_accuracy: 0.1565 - val_loss: 2.7458
Epoch 2/20
33/33    12s 286ms/step - accuracy: 0.1396 - loss: 2.6565 - val_accuracy: 0.1794 - val_loss: 2.4508
Epoch 3/20
33/33    12s 339ms/step - accuracy: 0.2875 - loss: 2.3504 - val_accuracy: 0.3931 - val_loss: 2.1184
Epoch 4/20
33/33    19s 280ms/step - accuracy: 0.4659 - loss: 1.6888 - val_accuracy: 0.4427 - val_loss: 1.8842
Epoch 5/20
33/33    7s 285ms/step - accuracy: 0.6033 - loss: 1.3241 - val_accuracy: 0.5153 - val_loss: 1.6158
Epoch 6/20
33/33    11s 223ms/step - accuracy: 0.7052 - loss: 0.9721 - val_accuracy: 0.5687 - val_loss: 1.5727
Epoch 7/20
33/33    8s 242ms/step - accuracy: 0.7544 - loss: 0.7184 - val_accuracy: 0.6450 - val_loss: 1.4110
Epoch 8/20
33/33    9s 203ms/step - accuracy: 0.8651 - loss: 0.4332 - val_accuracy: 0.6450 - val_loss: 1.4014
Epoch 9/20
33/33    12s 244ms/step - accuracy: 0.9262 - loss: 0.2956 - val_accuracy: 0.6565 - val_loss: 1.4581
Epoch 10/20
33/33    9s 264ms/step - accuracy: 0.9333 - loss: 0.2224 - val_accuracy: 0.6679 - val_loss: 1.6836
Epoch 11/20
33/33    8s 228ms/step - accuracy: 0.9513 - loss: 0.1682 - val_accuracy: 0.6603 - val_loss: 1.6352
Epoch 12/20
33/33    9s 275ms/step - accuracy: 0.9712 - loss: 0.1147 - val_accuracy: 0.7023 - val_loss: 1.5637
Epoch 13/20
33/33    8s 201ms/step - accuracy: 0.9864 - loss: 0.0527 - val_accuracy: 0.6832 - val_loss: 1.8641
Epoch 14/20
33/33    12s 241ms/step - accuracy: 0.9994 - loss: 0.0252 - val_accuracy: 0.7061 - val_loss: 1.8748
Epoch 15/20
33/33    11s 279ms/step - accuracy: 0.9927 - loss: 0.0401 - val_accuracy: 0.6870 - val_loss: 1.8841
Epoch 16/20
33/33    7s 214ms/step - accuracy: 0.9690 - loss: 0.1187 - val_accuracy: 0.7023 - val_loss: 1.5193
Epoch 17/20
33/33    11s 250ms/step - accuracy: 0.9946 - loss: 0.0329 - val_accuracy: 0.7099 - val_loss: 1.6269
Epoch 18/20
33/33    11s 280ms/step - accuracy: 0.9987 - loss: 0.0111 - val_accuracy: 0.6870 - val_loss: 1.9409
Epoch 19/20
33/33    10s 297ms/step - accuracy: 1.0000 - loss: 0.0074 - val_accuracy: 0.7366 - val_loss: 1.8693
Epoch 20/20
33/33    9s 266ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.7214 - val_loss: 1.9205

```

Se aprecia un alto nivel de precio obtenido a través del modelo entrenado.

```

[14] # Evaluar el modelo
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'\nPrecisión en el conjunto de prueba: {test_acc}')

1s [14] 11/11 - 1s - 86ms/step - accuracy: 0.6911 - loss: 1.8868
Precisión en el conjunto de prueba: 0.6911314725875854

0s [14] # Guardar el modelo
# model.save('/content/drive/MyDrive/Personales/Estudio/BigData/Mod11/TareaFinal/face_recognition_model.h5')
model.save('/content/drive/MyDrive/ProyectoFotos/Modelo/face_recognition_model.h5')
print("Modelo y etiquetas guardados exitosamente.")

0s [14] WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We re
Modelo y etiquetas guardados exitosamente.

```

En el siguiente código - M11\_Final\_03RecFacial-verificar.ipynb - verificamos el modelo:

Cargamos el modelo entrenado para reconocimiento facial y las etiquetas asociadas.

```

▶ from tensorflow.keras.models import load_model
import json
import numpy as np

def load_labels(filename):
    with open(filename, 'r') as f:
        return json.load(f)

# Cargar el modelo y las etiquetas
model = load_model('/content/drive/MyDrive/ProyectoFotos/Modelo/face_recognition_model.h5')
label_mapping = load_labels('/content/drive/MyDrive/ProyectoFotos/JSON/face_recognition_model.json')

# Invertir el mapeo para obtener los nombres de las etiquetas
label_names = {v: k for k, v in label_mapping.items()}

```

La función identify\_person2, recibe como parámetro la imagen y devuelve como resultado la etiqueta asociada a la predicción si la confianza es mayor 0.85.

```

▶ def identify_person2(image_path):
    # Cargar la imagen
    img = cv2.imread(image_path)
    if img is None:
        print(f"No se pudo cargar la imagen: {image_path}")
        return None, 0

    # Preprocesar la imagen
    processed_img = preprocess_image(img, target_size=(IMG_SIZE, IMG_SIZE))

    if processed_img is None:
        print("No se pudo procesar la imagen.")
        return None, 0

    # Preparar la imagen para la predicción
    img_array = np.expand_dims(processed_img, axis=0)

    # Realizar la predicción
    prediction = model.predict(img_array) # Cambiado de img a img_array
    person_index = np.argmax(prediction)
    confidence = prediction[0][person_index]

    # Si la confianza es menor a 0.85, retornar None
    if confidence < 0.85:
        return None, confidence

    person_name = label_names[person_index]
    return person_name, confidence

```

Para realizar la técnica Fine Tuning tenemos el código M11\_Final\_04RecFacial-fine-tunning.ipynb.

```
▶ import numpy as np
import os
import cv2
import json
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

[ ] # Configuración
IMG_SIZE = 64
BATCH_SIZE = 32
EPOCHS = 20
LEARNING_RATE = 0.0001 # Tasa de aprendizaje más baja

[ ] # Función para cargar imágenes y etiquetas
def load_data(dir_path):
    images = []
    labels = []
    for person_name in os.listdir(dir_path):
        person_path = os.path.join(dir_path, person_name)
        if os.path.isdir(person_path):
            for img_name in os.listdir(person_path):
                img_path = os.path.join(person_path, img_name)
                img = cv2.imread(img_path)
                img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
                img = img[:, :, ::-1] / 255.0 # BGR to RGB and normalize
                images.append(img)
                labels.append(person_name)
    return np.array(images), np.array(labels)

[ ] from google.colab import drive
# Montar Google Drive
drive.mount('/content/drive')

Mounted at /content/drive
```

```

❶ # Cargar el modelo existente
model = load_model('/content/drive/MyDrive/ProyectoFotos/Modelo/face_recognition_model.h5')

# Cargar las etiquetas existentes
with open('/content/drive/MyDrive/ProyectoFotos/JSON/face_recognition_model.json', 'r') as f:
    existing_labels = json.load(f)

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` w

```

```

[ ] # Cargar una parte de los datos originales (ajusta la ruta según sea necesario)
original_dir_path = '/content/drive/MyDrive/ProyectoFotos/Celebrity_Faces_Dataset/personas_nuevas_224x224'
original_images, original_labels = load_data(original_dir_path)

[ ] # Cargar nuevos datos
new_dir_path = '/content/drive/MyDrive/ProyectoFotos/Celebrity_Faces_Dataset/personas_nuevas_224x224'
new_images, new_labels = load_data(new_dir_path)

[ ] # Combinar datos originales y nuevos
all_images = np.concatenate([original_images, new_images])
all_labels = np.concatenate([original_labels, new_labels])

[ ] # Actualizar el mapeo de etiquetas
all_unique_labels = list(set(list(existing_labels.keys()) + list(np.unique(all_labels))))
le = LabelEncoder()
le.fit(all_unique_labels)
label_mapping = dict(zip(le.classes_, le.transform(le.classes_)))

[ ] # Codificar todas las etiquetas
all_labels_encoded = le.transform(all_labels)
all_labels_one_hot = to_categorical(all_labels_encoded, num_classes=len(label_mapping))


```

```

[ ] # Modificar la capa de salida del modelo
model.pop() # Eliminar la última capa
for layer in model.layers[:-2]: # Congelar todas las capas excepto las dos últimas
    layer.trainable = False
new_output_layer = Dense(len(label_mapping), activation='softmax')
model.add(new_output_layer)

[ ] # Compilar el modelo actualizado con una tasa de aprendizaje más baja
optimizer = Adam(learning_rate=LEARNING_RATE)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

[ ] # Fine-tuning
history = model.fit(all_images, all_labels_one_hot, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_split=0.2)

Epoch 1/20
82/82 10s 107ms/step - accuracy: 0.1938 - loss: 2.6296 - val_accuracy: 0.1514 - val_loss: 2.4153
Epoch 2/20
82/82 7s 67ms/step - accuracy: 0.5602 - loss: 1.5674 - val_accuracy: 0.4159 - val_loss: 1.8029
Epoch 3/20
82/82 10s 67ms/step - accuracy: 0.6897 - loss: 1.2074 - val_accuracy: 0.5856 - val_loss: 1.4270
Epoch 4/20
82/82 11s 80ms/step - accuracy: 0.7845 - loss: 0.9696 - val_accuracy: 0.6774 - val_loss: 1.1979

```

```

[ ] def save_labels(labels, filename):
    labels = {k: int(v) for k, v in labels.items()}
    with open(filename, 'w') as f:
        json.dump(labels, f)

# Guardar el modelo y las etiquetas actualizadas
model.save('/content/drive/MyDrive/ProyectoFotos/Modelo/face_recognition_model_updated.h5')

def save_labels(labels, filename):
    labels = {k: int(v) for k, v in labels.items()}
    with open(filename, 'w') as f:
        json.dump(labels, f)

save_labels(label_mapping, '/content/drive/MyDrive/ProyectoFotos/JSON/face_recognition_model_updated.json')
print("Modelo y etiquetas actualizados guardados exitosamente.")

```

## Conclusiones

En el presente trabajo se ha demostrado la efectividad de las redes neuronales convolucionales(CNN) en el desarrollo de un sistema de reconocimiento facial que puede implementarse para la identificación de empleados dentro de una empresa o institución. A través de este sistema, se consigue la automatización del acceso a las instalaciones, desbloqueando puertas al reconocer el rostro del empleado, también el registro preciso de la hora de entrada y salida de cada persona.

Se exploró una técnica de preprocessamiento que ayuda a enfocar y ajustar las imágenes para la identificación de rostros solamente, permitiendo así optimizar posteriormente el modelo de entrenamiento.

Se definió un modelo de red neuronal convolucional a través de imágenes con sus correspondientes etiquetas en base a 17 personajes muy populares en el ámbito del cine, habiendo obtenido una alta precisión en el modelo luego de aplicarse 33 iteraciones. Posteriormente, se aplicó la verificación del modelo con una fotografía descargada de Internet de la actriz Megan Fox.

Se exploraron también alternativas de afinamiento (*fine tuning*) la cual puede utilizarse cuando no se dispone de muchas imágenes para el entrenamiento de una red neuronal.

Finalmente, se exploró y utilizó un microservicio de Pyngrok el cual adaptaría el trabajo realizado a la implementación de una aplicación en un dominio o de manera local.