

# Reconocimiento de Rostros con Transfer Learning

04 de noviembre de 2023

## 1. Introducción

El problema trata sobre crear un modelo que pueda reconocer mi propia cara utilizando un algoritmo pre-entrenado pero realizando un fine-tuning para específicamente mi cara. Para lograr esto lo que haremos será tomar los pesos base de MobileNetV2 quitando el clasificador y añadiendo el nuestro propio.

## 2. Sobre los datos

La red que queremos crear va a necesitar datos de caras clasificados en *Elias* y *not\_Elias* ya que queremos poder clasificar una imagen de mi cara. Es por esto que para obtener los datos utilizamos la base CelebA la cual contiene una gran cantidad de fotografías con caras de personas. <http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html>



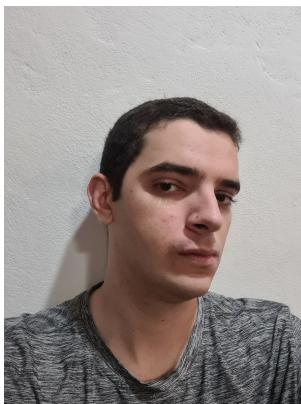
(a) Foto 1

(b) Foto 2

(c) Foto 3

Figura 1: Algunos ejemplos de las fotos de esta base.

Asimismo, tome una gran cantidad de fotos más para tener estas imágenes y lograr hacer la clasificación. No es una cantidad ideas ya que solo hay 300 fotografías de mi cara pero con algo de *data augmentation* esperamos que pueda funcionar el modelo.



(a) Foto 1



(b) Foto 2



(c) Foto 3

Figura 2: Algunos ejemplos de las fotos tomadas por mi.

### 3. Data Augmentation

Utilizamos el objeto `ImageDataGenerator` de tensorflow para aumentar nuestros datos. Particularmente, el siguiente código de python:

---

```
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,)
```

---

El cual refleja nuestras fotografías, las recorta y les hace transformaciones lineales para modificar la base directa de imágenes.

Cabe recalcar que no hubo aumento de datos para los datasets de validación y prueba.

### 4. Arquitectura de la red

Esta es una red clásica para realizar el reconocimiento de imágenes y es bastante simple cargarla. Después, nosotros agregamos 5 capas nuevas para obtener nuestro clasificador las cuales son:

- *GlobalAveragePooling2D*: para traducir nuestro resultado de MobileNetV2 a vectores unidimensionales con los que podamos trabajar.
- *Densa de 512 nodos*: Aquí empezamos la clasificación con una capa de 64 nodos con `batch_normalization` para intentar reducir algo el overfitting. La función de activación es una `relu`.
- *Dropout de 0.5*: De nuevo intentando atacar algo del overfitting dejamos de actualizar la capa anterior en el 50 % de los casos.
- *Densa de 128 nodos*: De nuevo continuamos con nuestra segunda capa de clasificación.

- *Densa de 1 nodo:* Esta será nuestro output el cual es binario por lo que tiene una función de activación sigmoide.

Para esta red se utilizó el optimizador Adam con una razón de aprendizaje de 0.0001. Con 15 epochas y un tamaño de batch de 30. El resumen de la red queda como sigue:

---

Model: "model\_8"

Layer (type)	Output Shape	Trainable Param #
<hr/>		
MobileNetV2	(None, 7, 7, 1280)	0
global_average_pooling2d_8	(None, 1280)	0
dense_18 (Dense)	(None, 512)	655872
dense_19 (Dense)	(None, 256)	65664
dense_20 (Dense)	(None, 1)	129
<hr/>		
Total params: 2979649 (11.37 MB)		
Trainable params: 721665 (2.75 MB)		
Non-trainable params: 2257984 (8.61 MB)		
<hr/>		

---

## 5. Resultados

Los resultados medianamente satisfactorios ya que nuestro conjunto de prueba obtuvo un total de 73 % de accuracy lo cual es bueno pero no excelente considerando que la mayoría de las fotografías de una clase eran bastante similares entre sí.

Sin embargo, sí logramos clasificar fotografías de forma correcta como en los siguientes ejemplos:

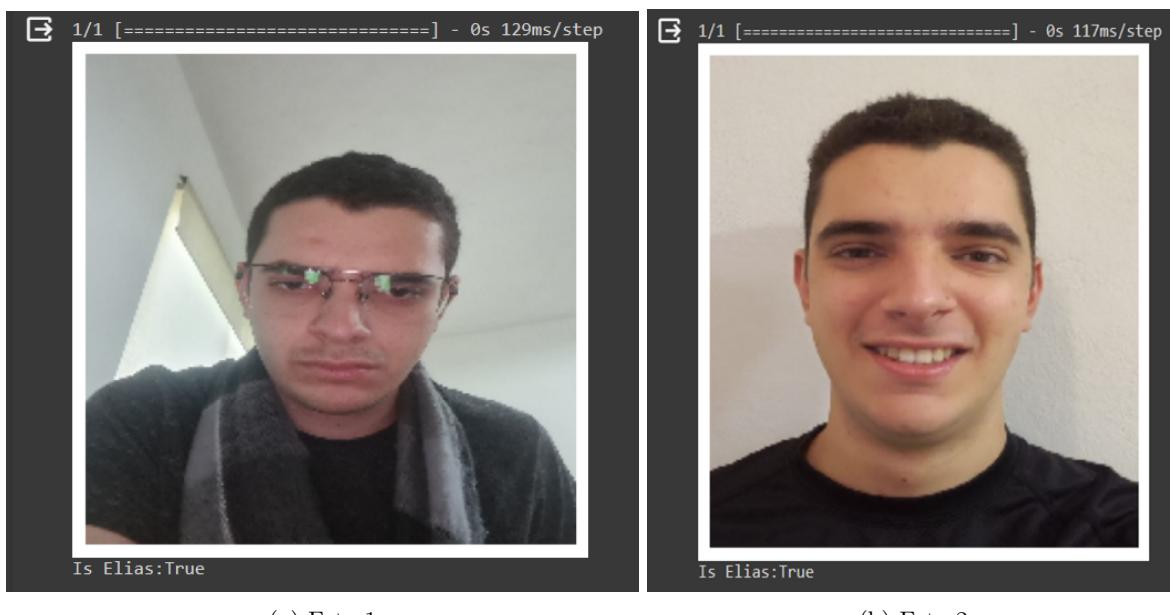


Figura 3: Ejemplos del modelo funcionando.

## 6. Posibles mejoras

Cabe recalcar que este modelo tiene grandes áreas de oportunidad. Principalmente que se eligió como base a MobileNetV2 debido a su peso ligero comparado con otras redes para que fuera rápido trabajar con esta pero utilizar un red más poderosa como los embeddings resultantes de la librería `face_recognition` como entrada a nuestro modelo clasificador puede mejorar el rendimiento.

## Collab

El código de este proyecto se puede encontrar en

<https://colab.research.google.com/drive/1AfefCL1Sx4pSsYsVsPWJTLYGRlSitGuY?usp=sharing>