UNIVERSIDAD POLITÉCNICA DE CARTAGENA (UPCT)

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIONES (ETSIT)

Grado en Ingeniería Telemática

APLICACIONES DE LA TECNOLOGÍA DE RECONOCIMIENTO FACIAL

Autor

Juan Guillermo Carrasco Martínez

Director

María Dolores Cano Baños

Agradecimientos

A mi madre y a mi hermana por aguantarme en toda mi trayectoria académica, a pesar de las dificultades.

A Lola, por abordar conmigo este proyecto y ayudarme en todo lo posible para que todo saliera bien.

Índice de contenidos

Relación de figuras

Capítulo 1. Introducción y Objetivos

1.1 Introducción

El objetivo de este Trabajo de Fin de Grado es la realización de un programa capaz de detectar la persona que aparece en una imagen y a partir de esa respuesta, realizar distintas acciones según el caso. Para llevarlo a cabo, me he apoyado en técnicas de inteligencia artificial, como son el aprendizaje automático y el profundo (más conocidos como: Machine Learning y Deep Learning respectivamente).

A lo largo de los siguientes capítulos, voy a explicar más en detalle en qué consiste mi proyecto, los resultados que he obtenido y algunos ejemplos de usos que se le podrían dar en un futuro con muy pocos cambios. Además, antes de entrar en los detalles del proyecto, es necesario exponer algunos conceptos teóricos para entender mejor la temática.

1.2 Justificación del proyecto

Podemos justificar este proyecto gracias al auge que está teniendo la Inteligencia Artificial (IA) en muchos aspectos de la vida cotidiana en los últimos años. La IA se utiliza sobre todo para tomar decisiones complejas basadas en datos, que los humanos o bien no serían capaces de tomar o, si lo hacen, tardarían muchísimo tiempo.

El concepto de IA no es nada nuevo, se remonta a la época de los 50 del siglo pasado. Tras las pruebas que realizó el matemático Allan Turing, quien es considerado como uno de "los padres de la tecnología de la información" en su artículo "Computing machinery and intelligence", no fue hasta 1956 cuando nació el concepto de IA gracias a John McCarthy, Marvin Minsky y Claude Shannon. Éstos definieron la IA como "la ciencia e ingeniería de hacer máquinas inteligentes". Pero la verdadera revolución en términos de IA a un nivel práctico, no se produjo hasta que los ordenadores fueron capaces de procesar una cantidad enorme de datos en un tiempo aceptable para así poder experimentar con la IA de una manera más eficiente y rápida.

Hoy en día, tenemos muchos ejemplos en los que la IA nos puede facilitar la vida, como son:

- Vehículos autónomos.

- Asistentes digitales o chatbots en páginas web para atender todas nuestras cuestiones.

- Traducciones en tiempo real en vídeos de YouTube.

- Asistentes de voz como Google Home o Alexa.

- Recomendaciones de nuestros gustos en aplicaciones como Spotify o Netflix.

- Filtros de SPAM en servicios de correo electrónico, como Gmail y Outlook.

- Reconocimiento facial usado para desbloquear nuestros smartphones.

En definitiva, este proyecto me va a ayudar a sentar las bases en un paradigma que está todavía por terminar de explorar y que gracias a él sin duda los seres humanos vamos a ser capaces de realizar ciertas cosas que en principio nos parecerían impensables.

1.3 Objetivos

Desde un punto de vista más amplio, el objetivo primordial de este proyecto es iniciarse en la inteligencia artificial mediante la investigación de los principales algoritmos de aprendizaje profundo, en concreto las Redes Neuronales Convolucionales.

El objetivo específico de este proyecto es:

- Crear una aplicación capaz de identificar a la persona que se sienta delante de un ordenador y a partir de esta identificación, mostrar mensajes emergentes en pantalla.

Además de los objetivos comentados cabe destacar algunos otros como:

- Aprendizaje del lenguaje Python y el uso de librerías como Tensorflow y Keras para ayudar a implementar los algoritmos.

- Uso de las Redes Neuronales Convolucionales (CNN) para la clasificación de imágenes.

- Interpretación y uso de los resultados que nos proporciona una CNN para diferentes tareas, en concreto para reconocer a la persona que aparece en una imagen.

- Saber encender la cámara del ordenador y guardar la imagen que captura ésta en una carpeta del sistema de archivos mediante el lenguaje de programación C#.

- Mostrar en pantalla un mensaje emergente mediante el lenguaje de programación C#.

1.4 Contenido del resto de la memoria

A continuación, voy a definir el contenido de los siguientes capítulos de la memoria:

- En el capítulo 2 voy a exponer la base teórica necesaria para entender el desarrollo del proyecto.

- En el capítulo 3 voy a explicar el código de mi aplicación en profundidad, para que se puedan apreciar todos los detalles que he incluido. Además, también voy a incluir su funcionamiento.

- En el capítulo 4 voy a detallar los resultados obtenidos al ejecutar mi aplicación.

- En el capítulo 5 voy a relatar las conclusiones obtenidas una vez terminado el TFG, así como también algunos ejemplos de usos que se le podrían dar en un futuro y los conocimientos que he aprendido.

Capítulo 2. Base Teórica

2.1 Conceptos teóricos

Antes de explicar mi proyecto, vamos a introducir unos conceptos previos necesarios para entenderlo:

- Inteligencia Artificial

  "Capacidad de una máquina de imitar las funciones cognitivas que hasta ahora se asociaban exclusivamente a los humanos. Percibir, razonar, aprender o solucionar problemas, son algunas de las cosas que puede hacer una inteligencia artificial". Aunque todavía al escuchar este término nos recuerda a una película de ciencia ficción, esta tecnología ya está suplida en nuestra vida cotidiana. Algunos ejemplos de inteligencia artificial serían: vehículos autónomos, robots de asistencia, reconocimiento de imágenes, etc.

- Machine Learning

  "Rama de la inteligencia artificial que hace posible que las máquinas aprendan por sí mismas, sin depender de órdenes. Es decir, el Machine Learning implica que la máquina es entrenada para automatizar tareas imposibles para un ser humano y, como resultado de ese aprendizaje, puede realizar predicciones". Esta innovación les da a los ordenadores el poder de resolver las cosas sin que las tengan programadas explícitamente, simplemente con tener los datos de entrenamiento el algoritmo es capaz de construir su propia lógica. Algunos ejemplos de Machine Learning incluyen: recomendaciones de Netflix o incluso respuestas automatizadas en servicios de correo electrónico como Gmail.

- Deep Learning

  "Lo podemos definir como un tipo de Machine Learning, solo que algo más complejo. El Deep Learning o aprendizaje profundo es un conjunto de algoritmos que imita las redes neuronales de un cerebro humano". En esta técnica, el algoritmo aprende por sí mismo, pero por capas. La profundidad del modelo dependerá del número de etapas. Algunos ejemplos de Deep Learning son: los traductores inteligentes y los asistentes de voz como Siri o Google Home [1].

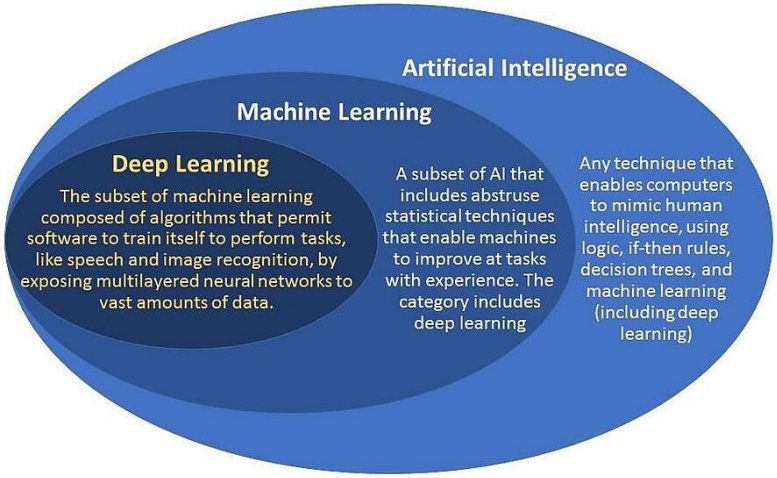

Figura 1. Conceptos clave [2]

2.2 Introducción a algoritmos de Machine Learning

Tenemos tres tipos fundamentales de algoritmos de Machine Learning:

- Aprendizaje supervisado

  "Se basa en modelos predictivos que hacen uso de datos de entrenamiento. Dado un conjunto conocido de datos, se pretende que el sistema sea capaz de lograr una determinada salida, de forma que el modelo es entrenado hasta lograr resultados adecuados". Los principales algoritmos son: árboles de decisión, "clasificaciones Naïve Bayes", "regresión ordinaria por mínimos cuadrados", "regresión logística" y "Support Vector Machines (SVM)".

- Aprendizaje no supervisado

  "Algoritmos similares a los de aprendizaje supervisado, pero estos ajustan su modelo únicamente en función de los datos de entrada. Dicho de un modo sencillo, el algoritmo realiza un auto entrenamiento sin indicaciones externas". Los principales algoritmos de este aprendizaje son: "algoritmos de agrupamiento (clustering)", "análisis de componentes principales (PCA)", "Singular Value Decompotition (SVD)" y "análisis de componentes independientes (ICA)".

- Aprendizaje por refuerzo

  "Consiste en la iteración constante y basada en "prueba y error" que una máquina es capaz de realizar ante determinadas condiciones o entorno dado (por ejemplo, las reglas de un juego)" y con un objetivo específico llamado "recompensa". De esta forma se pueden obtener resultados, patrones y conclusiones basados en experiencia previa generada por el propio algoritmo. Un ejemplo de este modelo de aprendizaje ha sido la IA ajedrecista "AlphaZero de DeepMind" [3].

En este proyecto me voy a centrar en el aprendizaje supervisado, ya que la funcionalidad del programa está basada en una Red Neuronal Convolucional a la que le pasamos los datos de entrenamiento; es decir, entrenamos la red hasta lograr los resultados adecuados.

2.3 Redes Neuronales Convolucionales (CNN)

“Las Redes Neuronales Artificiales (RNA) son un modelo computacional evolucionado a partir de diversas aportaciones científicas. Consiste en un conjunto de unidades, llamadas neuronas artificiales, conectadas entre sí para transmitirse señales”.

“Una Red Neuronal Convolucional (CNN) es un tipo de RNA con aprendizaje supervisado que procesa sus capas imitando al córtex visual del ojo humano para identificar distintas características en las entradas que en definitiva hacen que pueda identificar objetos”. Para poder realizar eso, la CNN contiene varias etapas ocultas especializadas y con una jerarquía: con esto intenta explicar que las primeras etapas pueden detectar líneas o incluso curvas y se van especializando hasta llegar a etapas más profundas que reconocen formas complejas como un rostro u objetos muy específicos.

Seguidamente, voy a explicar en detalle cada una de las capas por las que está compuesta una CNN.

En la siguiente ilustración (Figura 2), podemos observar la estructura típica de una CNN con sus capas principales.

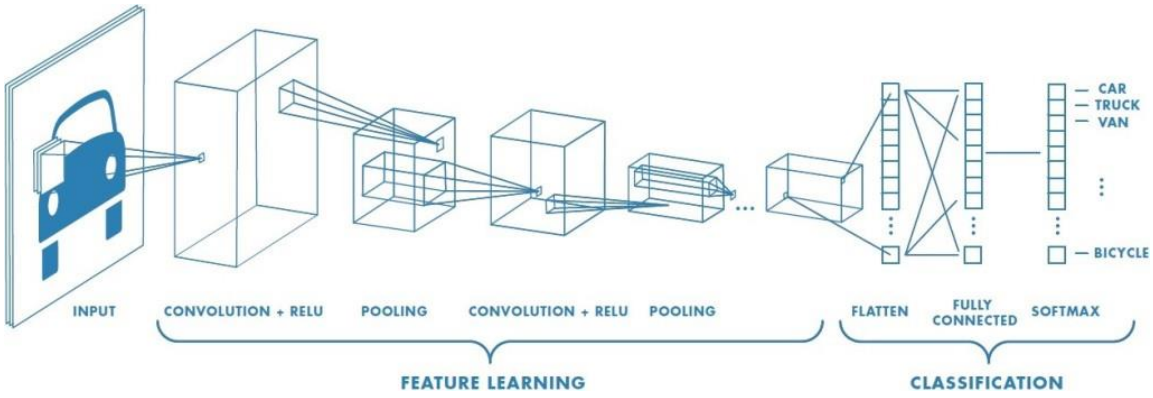

Figura 2. Estructura de una CNN [4]

Las capas en las que voy a centrar mi explicación son:

- **Capa de Convolución**
- **Capa de Pooling**
- **Capa totalmente conectada**
- **Capa de salida**

A continuación, voy a explicar cada una de ellas de forma detallada.

- **Convolución**

  A la imagen de entrada, antes de pasarla por la capa de convolución hay que realizarle un preprocesamiento; es decir, cada imagen tiene unos valores de píxeles que van desde 0 a 255, pues este preprocesamiento consiste en una transformación de cada valor de píxel: "valor/255" para poder tener un valor normalizado entre 0 y 1. Una vez hecho esto, pasamos a la capa de convolución.

  La función principal de esta capa es la operación matemática de la convolución, la cual consiste en tomar "grupos de píxeles cercanos" de una imagen e ir operando (producto escalar) contra una pequeña matriz llamada kernel. Esta matriz (por ejemplo, de 2x2 píxeles) recorre la imagen de entrada (de izquierda a derecha y de arriba a abajo) para generar una matriz de salida. No tenemos un solo kernel, si no que podemos tener muchos ("su conjunto se llama filtros"). Por ejemplo, podemos tener 64 filtros, con lo que tendríamos 64 matrices de salida o "imágenes filtradas nuevas". Estas imágenes están "dibujando" ciertas características de la imagen de entrada. Una vez tenemos todas estas "imágenes nuevas", a cada una de ellas le aplicamos la función de activación **"ReLu"**, la cual consiste en poner a 0 cero los números negativos y dejar como están los positivos.

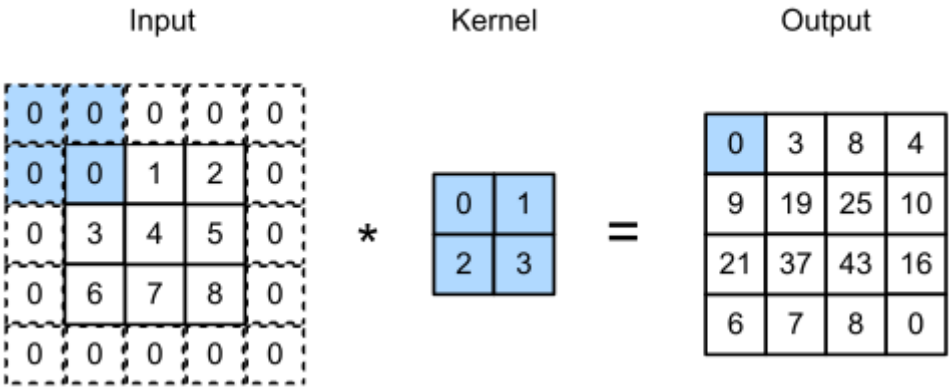

Figura 3. Capa de Convolución [5]

- **Pooling**

  En esta capa realizamos una reducción de la cantidad de neuronas antes de efectuar una nueva convolución. Para poder alcanzar eso, realizaremos un proceso de "subsampling" en el que disminuimos el tamaño de las imágenes filtradas, pero donde "**deberán prevalecer las características más importantes que detectó cada filtro**". Hay diversos tipos, pero yo me voy a centrar en el "más usado: **Max-Pooling**".

Para explicarlo, vamos a utilizar un ejemplo: suponemos que tenemos un Max-Pooling de 2x2; es decir, recorremos cada una de las imágenes obtenidas en la capa de convolución tomando "2x2 píxeles (2 de alto por 2 de ancho = 4 píxeles") y vamos preservando el valor "más alto" de entre esos 4 píxeles. Usando un tamaño de 2x2, la imagen que obtenemos estará reducida "a la mitad".

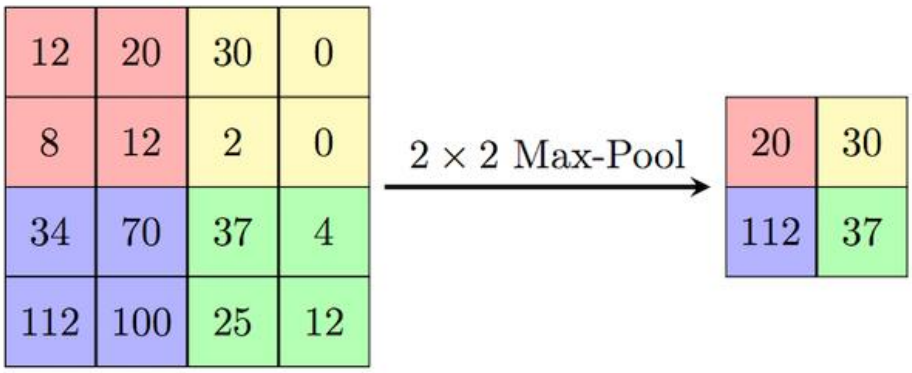

Figura 4. Max-Pooling [6]

Estas dos capas de convolución y pooling se repiten varias veces según el problema que queramos resolver haciendo uso de una CNN.

- **Capa totalmente conectada**

  Por último, vamos a coger la última capa a la que le aplicamos el Max-Pooling, que es "tridimensional" (alto, ancho, 3 canales RGB si la imagen es en color o 1 si es en blanco y negro) y la "aplanamos" para que pase a ser "una capa de neuronas tradicionales". Por ejemplo, podríamos aplanarla y también conectarla a una capa de 256 neuronas totalmente conectada en la que todas las neuronas están conectadas con las de la capa anterior.

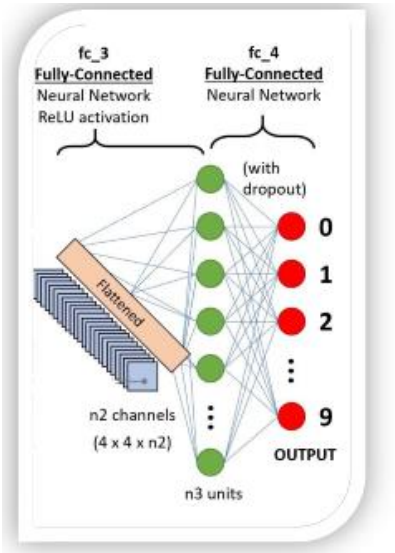

Figura 5. Capa totalmente conectada [7]

- **Capa de salida**

  Como paso final, la salida de la capa totalmente conectada la conectamos con una capa de salida final que va a tener el mismo número de neuronas que las clases que estamos clasificando en ese caso. Esta capa tiene la función de activación **"Softmax"**, la cual se encarga de pasar a probabilidad los valores de las neuronas de salida. Por ejemplo, una salida [0,3 0,7] nos indica un 30% de probabilidad de que en la imagen aparezca una persona y un 70% de que aparezca otra distinta [8].

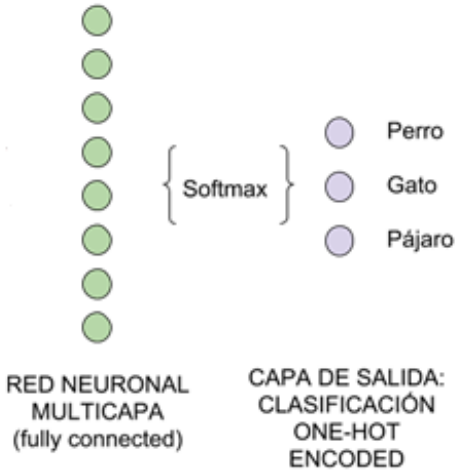

Figura 6. Capa de salida

Capítulo 3. Planteamiento del problema, diseño y desarrollo de la aplicación propuesta

3.1 Conjunto de datos

Para poder llevar a cabo este proyecto, he recopilado en total 238 imágenes haciendo fotos con mi ordenador personal. Éstas, las he dividido en un conjunto de datos de entrenamiento compuesto por 193 imágenes de las cuales en 116 aparezco yo y en 77 aparece mi madre. Las 45 imágenes restantes se reservan como datos de validación de las cuales en 25 vuelvo a aparecer yo y en 20 aparece mi madre.

En las siguientes figuras voy a mostrar un ejemplo de las fotos del dataset, en la primera las imágenes pertenecen a mí mismo (Figura 7) y en la segunda pertenecen a mi madre (Figura 8).

Figura 7. Ejemplos de fotos de entrenamiento 1

Figura 8. Ejemplos de fotos de entrenamiento 2

3.2 Entrenamiento del modelo

Para ejecutar el entrenamiento y predicción de este modelo, hemos utilizado la herramienta Anaconda Navigator. “Anaconda es una suite de código abierto que engloba una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la ciencia de datos con Python y, además, te permite iniciar aplicaciones y administrar fácilmente paquetes y entornos sin usar comandos”.

El código utilizado para el entrenamiento y la predicción del modelo se ha elaborado haciendo uso de un “Jupyter Notebook”, ya que así es posible crear bloques del programa en Python e irlos ejecutando bloque a bloque de forma interactiva.

El código completo de mi aplicación se adjunta en el anexo de este documento.

A continuación, voy a explicar con detalle el código del archivo **Entrenar.ipynb**.

Al principio del código he importado las librerías necesarias para llevar a cabo este proyecto: “os”, “sys”, “keras” y “matplotlib”. “os” y “sys” para movernos por carpetas dentro del sistema operativo, “keras” para definir la arquitectura de la red y “matplotlib” para ayudarme a generar gráficas.

```
import os
import sys
import keras
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.python.keras.layers import Convolution2D, MaxPooling2D
from tensorflow.python.keras import backend as K
```

Figura 9. Entrenar.ipynb fragmento 1

A continuación, he utilizado la variable “K” declarada más arriba para que en caso de que haya una sesión de keras corriendo en segundo plano al empezar el entrenamiento, cerrarla. También he definido dos variables que contienen la ruta donde se encuentran las imágenes de entrenamiento y validación.

```
K.clear_session()
```

```
data_entrenamiento = 'C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Archivos\\Fotos_TFG\\Entrenamiento'
data_validacion = 'C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Archivos\\Fotos_TFG\\Validacion'
```

Figura 10. Entrenar.ipynb fragmento 2

Como siguiente paso, he definido algunos parámetros necesarios para el correcto funcionamiento del programa:

```
#Parametros

epochs = 100
altura, longitud = 48, 48
batch_size = 32
filtrosConv1 = 32
filtrosConv2 = 64
tamano_filtro1 = (3,3)
tamano_filtro2 = (3,3)
tamano_pool = (2,2)
clases = 2
lr = 0.0005
```

Figura 11. Entrenar.ipynb fragmento 3

- **Epochs**: Número de fases de entrenamiento.

- **Altura, longitud**: Tamaño de la imagen en píxeles.

- **Batch_size**: Número de imágenes que se procesarán en cada fase de entrenamiento.

- **FiltrosConv1 y FiltrosConv2**: Número de filtros (kernels) utilizados en la primera y segunda convolución respectivamente.

- **Tamano_filtro1 y Tamano_filtro2**: Tamaño en píxeles de los filtros (kernels) utilizados en la primera y segunda convolución respectivamente.

- **Tamano_pool**: Tamaño en píxeles del filtro utilizado para realizar el Max_Pooling.

- **Clases**: Número de clases en las que divido las imágenes del modelo.

- **Lr**: Tasa de aprendizaje del optimizador utilizado en el modelo.

Como tenemos pocas muestras, el siguiente paso es realizar un aumento de las imágenes de entrenamiento a través de una serie de transformaciones aleatorias y así evitar el sobreajuste.

```
#Preprocesamiento de imagenes

entrenamiento_datagen = ImageDataGenerator(
    rescale = 1./255,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    shear_range = 0.3,
    zoom_range = 0.3,
    horizontal_flip = True,
)

validacion_datagen = ImageDataGenerator(
    rescale = 1./255
)
```

Figura 12. Entrenar.ipynb fragmento 4

- Escalamos las imágenes tanto las de entrenamiento como las de validación a un factor 1/255 con **"rescale"**.

- Realizamos desplazamientos horizontales a las imágenes de entrenamiento con **"width_shift_range"**.

- Realizamos desplazamientos verticales a las imágenes de entrenamiento con **"height_shift_range"**.

- Inclinamos nuestras imágenes de entrenamiento aleatoriamente con "**shear_range**".

- Hacemos zoom a las imágenes de entrenamiento de forma aleatoria con **"zoom_range"**.

- Invertimos las imágenes de entrenamiento aleatoriamente con **"horizontal_flip"**.

A continuación, entramos en el directorio donde tenemos las imágenes almacenadas y generamos los lotes de imágenes con las modificaciones correspondientes. Añadimos el parámetro **"class_mode = categorical"** para indicar que la clasificación de las imágenes es categórica.

```
imagen_entrenamiento = entrenamiento_datagen.flow_from_directory(
    data_entrenamiento,
    target_size = (altura, longitud),
    batch_size = batch_size,
    class_mode = 'categorical'
)

imagen_validacion = validacion_datagen.flow_from_directory(
    data_validacion,
    target_size = (altura, longitud),
    batch_size = batch_size,
    class_mode = 'categorical'
)
```

Figura 13. Entrenar.ipynb fragmento 5

El siguiente paso, una vez tenemos todos los parámetros establecidos y las imágenes cargadas, es construir una Red Neuronal Convolucional (CNN) con la que poder clasificar nuestras imágenes. Mi CNN está compuesta por las siguientes capas:

- 2 capas de convolución con los parámetros establecidos al principio del código, función de relleno **"same"** para que la imagen conserve su tamaño original y función de activación **"ReLu"**. Cada una de ellas con su correspondiente capa de **Max-Pooling** de tamaño 2x2.
- Una capa para "aplanar" el resultado de la última capa de Max-Pooling.
- Una capa densa totalmente conectada compuesta por 256 neuronas y función de activación **"ReLu"**.
- Otra capa densa totalmente conectada compuesta por el mismo número de neuronas que de clases tenemos en el modelo y función de activación **"Softmax"**.

```
#Crear la Red Neuronal Convolucional (CNN)

cnn = Sequential()

cnn.add(Convolution2D(filtrosConv1, tamano_filtro1, padding='same', input_shape=(altura, longitud, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size = tamano_pool))

cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding='same', activation='relu'))
cnn.add(MaxPooling2D(pool_size = tamano_pool))

cnn.add(Flatten())
cnn.add(Dense(256, activation = 'relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases, activation = 'softmax'))
```

Figura 14. Entrenar.ipynb fragmento 6

Ahora definimos algunos parámetros necesarios para optimizar nuestra CNN.

```
cnn.compile(loss = 'categorical_crossentropy', optimizer = keras.optimizers.Adam(lr=lr), metrics=['accuracy'])
```

Figura 15. Entrenar.ipynb fragmento 7

Antes de empezar el entrenamiento de nuestro modelo, hemos definido dos devoluciones de llamada **"callbacks"**:

- **EarlyStopping:** Finaliza el entrenamiento cuando las pérdidas de validación (**"val_loss"**) ya no mejoran más.
- **ReduceLROnPlateau:** Reduce la tasa de aprendizaje cuando la precisión (**"accuracy"**) ya no mejora más.

Una vez definidas las devoluciones de llamada, entrenamos el modelo utilizando **"cnn.fit"** y guardamos los resultados en la variable H para su posterior representación gráfica.

```
early_stopping = EarlyStopping(
    monitor = 'val_loss',
    min_delta = 0,
    patience = 12,
    verbose = 1,
    mode = 'auto',
    restore_best_weights = True
)

reduce_lr = ReduceLROnPlateau(
    monitor = 'accuracy',
    min_delta = 0,
    factor = 0.2,
    patience = 5,
    min_lr = 1e-7,
    verbose = 1
)

callbacks = [early_stopping, reduce_lr]

H = cnn.fit(imagen_entrenamiento, epochs = epochs, validation_data = imagen_validacion, callbacks = callbacks)
```

Figura 16. Entrenar.ipynb fragmento 8

Las rutas en las que se guardan el modelo y sus correspondientes pesos las podemos observar en la siguiente imagen (Figura 17).

```
dir = 'C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Modelo'

if not os.path.exists(dir):
    os.mkdir(dir)
cnn.save('C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Modelo\\modelo.h5')
cnn.save_weights('C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Modelo\\pesos.h5')
```

Figura 17. Entrenar.ipynb fragmento 9

```
Epoch 30/100
7/7 [==============================] - ETA: 0s - loss: 0.1912 - accuracy: 0.9585
Epoch 00030: ReduceLROnPlateau reducing learning rate to 4.000000262749381e-06.
7/7 [==============================] - 1s 169ms/step - loss: 0.1912 - accuracy: 0.9585 - val_loss: 0.0698 - val_accuracy: 1.000
0
Epoch 31/100
7/7 [==============================] - 1s 166ms/step - loss: 0.2462 - accuracy: 0.9378 - val_loss: 0.0700 - val_accuracy: 1.000
0
Epoch 32/100
7/7 [==============================] - 1s 174ms/step - loss: 0.2253 - accuracy: 0.9275 - val_loss: 0.0714 - val_accuracy: 1.000
0
Epoch 33/100
7/7 [==============================] - 1s 170ms/step - loss: 0.2068 - accuracy: 0.9326 - val_loss: 0.0727 - val_accuracy: 1.000
0
Epoch 34/100
7/7 [==============================] - 1s 204ms/step - loss: 0.2656 - accuracy: 0.9067 - val_loss: 0.0730 - val_accuracy: 1.000
0
Epoch 35/100
7/7 [==============================] - ETA: 0s - loss: 0.2250 - accuracy: 0.9171
Epoch 00035: ReduceLROnPlateau reducing learning rate to 8.000000889296644e-07.
7/7 [==============================] - 1s 204ms/step - loss: 0.2250 - accuracy: 0.9171 - val_loss: 0.0713 - val_accuracy: 1.000
0
Epoch 36/100
7/7 [==============================] - 1s 181ms/step - loss: 0.2397 - accuracy: 0.9016 - val_loss: 0.0712 - val_accuracy: 1.000
0
Epoch 37/100
7/7 [==============================] - 1s 170ms/step - loss: 0.1853 - accuracy: 0.9534 - val_loss: 0.0712 - val_accuracy: 1.000
0
Epoch 38/100
7/7 [==============================] - 1s 173ms/step - loss: 0.2347 - accuracy: 0.9119 - val_loss: 0.0711 - val_accuracy: 1.000
0
Epoch 39/100
7/7 [==============================] - 1s 180ms/step - loss: 0.2644 - accuracy: 0.8912 - val_loss: 0.0710 - val_accuracy: 1.000
0
Epoch 40/100
7/7 [==============================] - ETA: 0s - loss: 0.2489 - accuracy: 0.9119
Epoch 00040: ReduceLROnPlateau reducing learning rate to 1.6000001323845936e-07.
7/7 [==============================] - 2s 226ms/step - loss: 0.2489 - accuracy: 0.9119 - val_loss: 0.0710 - val_accuracy: 1.000
0
Epoch 41/100
7/7 [==============================] - 1s 183ms/step - loss: 0.2198 - accuracy: 0.9119 - val_loss: 0.0710 - val_accuracy: 1.000
0
Epoch 42/100
7/7 [==============================] - ETA: 0s - loss: 0.2431 - accuracy: 0.9067Restoring model weights from the end of the bes
t epoch.
7/7 [==============================] - 1s 190ms/step - loss: 0.2431 - accuracy: 0.9067 - val_loss: 0.0710 - val_accuracy: 1.000
0
Epoch 00042: early stopping
```

Figura 18. Salida del entrenamiento

Como podemos observar en la imagen anterior (Figura 18), el mejor valor de pérdidas de validación (**"val_loss"**) se alcanza en la fase 30 y se corresponde con 0,0698. También en esa fase se alcanza la mejor precisión de validación (**"val_accuracy"**) de todo el entrenamiento con un valor del 100%.

Para terminar con el entrenamiento del modelo, hemos representado dos gráficas. La de la derecha (Figura 19 b) compara la precisión de entrenamiento con la de validación y la de la izquierda (Figura 19 a) hace lo mismo con las pérdidas de entrenamiento y validación.

En las gráficas (Figura 19 a y b) podemos apreciar como la precisión de validación se estabiliza a partir de la fase 15 de entrenamiento y las pérdidas de validación lo hacen en la fase 30 con unos resultados bastante buenos. Otro aspecto por destacar es que, aunque hemos establecido 100 fases de entrenamiento, el algoritmo termina en la 42, esto es a consecuencia de haber fijado una parada anticipada cuando las pérdidas de validación dejaran de mejorar.

```
plt.figure(figsize=(18,6))

plt.subplot(1,2,1)
plt.plot(H.history['loss'], color = "blue", label = "Training")
plt.plot(H.history['val_loss'], color = "orange", label = "Validation")
plt.title("Training Loss vs Validation Loss")
plt.ylabel("Loss")
plt.xlabel("Number of Epochs")
plt.legend()
```

```
plt.figure(figsize=(18,6))

plt.subplot(1,2,1)
plt.plot(H.history['accuracy'], color = "blue", label = "Training")
plt.plot(H.history['val_accuracy'], color = "orange", label = "Validation")
plt.title("Training Accuracy vs Validation Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Number of Epochs")
plt.legend()
```

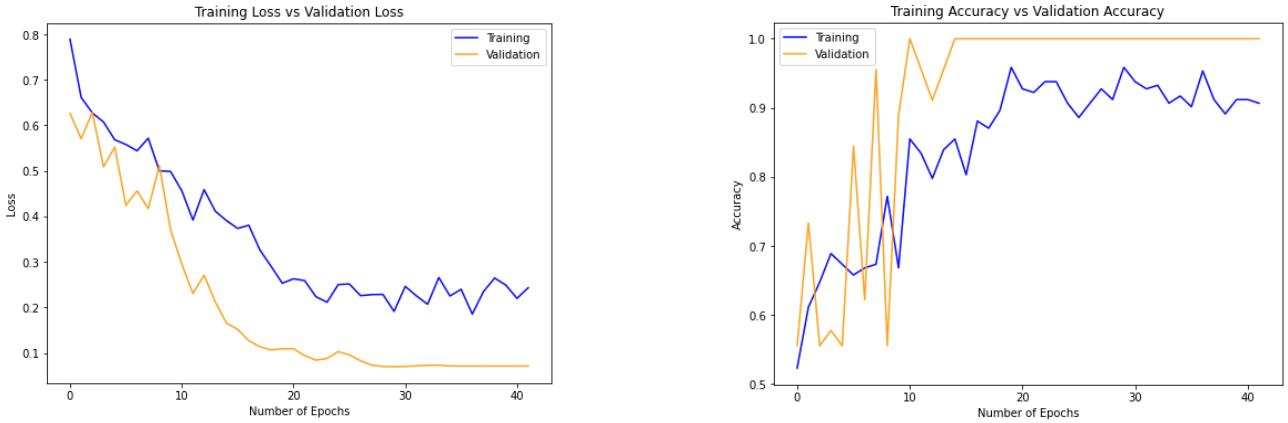

Figura 19. Gráficas de pérdidas y precisión del modelo

3.3 Predicción del modelo

Una vez explicado el código utilizado para entrenar mi modelo, voy a pasar a explicar el archivo encargado de hacer predicciones sobre imágenes utilizando los datos entrenados por mi Red Neuronal Convolucional que recibe el nombre de **Predecir.ipynb**.

Al principio del código he importado las librerías necesarias para este archivo: “numpy”, “time”, “keras” y “watchdog”. “numpy” para trabajar con las imágenes, “time” para poder utilizar la función “sleep”, “keras” para importar funciones necesarias y “watchdog” para monitorizar eventos.

Antes de efectuar las operaciones necesarias para realizar predicciones, también he definido algunos parámetros necesarios para su correcto funcionamiento como son:

- El tamaño de las imágenes en píxeles.
- Dos variables que indican la ruta donde están guardados tanto el modelo entrenado en el archivo Entrenar.ipynb como sus correspondientes pesos.
- Una variable llamada “cnn” donde cargamos el modelo gracias a la función **“load_model”** de keras.
- A la variable “cnn” le cargamos sus correspondientes pesos gracias a la función **“load_weights”**.

```
import numpy as np
import time
from keras.preprocessing.image import load_img, img_to_array
from keras.models import load_model
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

longitud, altura = 48, 48
modelo ='C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Modelo\\modelo.h5'
pesos ='C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Modelo\\pesos.h5'
cnn = load_model(modelo)
cnn.load_weights(pesos)
```

Figura 20. Predecir.ipynb fragmento 1

Seguidamente, he definido una función llamada **"predict"** a la cual le pasas la ruta en la que se guarda una imagen y ésta te devuelve el nombre de la persona que aparece en ella y, además, según esa respuesta, escribe en el fichero **"Respuesta_CNN.txt"** un **"0"** si en la foto aparece mi madre o un **"1"** si aparezco yo para que después el programa encargado de hacer las fotos con la cámara del ordenador sepa que persona aparece y pueda actuar en consecuencia.

```
def predict(file_path):
    time.sleep(1)
    x = load_img(file_path, target_size=(longitud, altura))
    x = img_to_array(x)
    x = np.expand_dims(x, axis=0)
    arreglo = cnn.predict(x)
    resultado = arreglo[0]
    respuesta = np.argmax(resultado)
    if respuesta == 0:
        print('Antonia Martínez Navarro')
        archivo = open('C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Archivos\\CNN\\Respuesta_CNN.txt', 'w')
        archivo.write('0')
        archivo.close()
    elif respuesta == 1:
        print('Juan Guillermo Carrasco Martínez')
        archivo = open('C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Archivos\\CNN\\Respuesta_CNN.txt', 'w')
        archivo.write('1')
        archivo.close()
    return respuesta
```

Figura 21. Predecir.ipynb fragmento 2

Por último, he definido un manejador de eventos que proviene de la librería "watchdog" de Python que monitoriza continuamente el directorio donde se guarda la imagen y cuando se archiva una imagen nueva o se modifica la que ya había llama a la función **"predict"** definida más arriba.

```
class MyEventHandler(FileSystemEventHandler):
    def on_any_event(self, event):
        predict(r'C:\Users\Usuario\Desktop\Curso 2021-22\TFG\Archivos\Fotos_TFG\Fotos_camara\Foto.jpg')

observer = Observer()
observer.schedule(MyEventHandler(), "C:\\Users\\Usuario\\Desktop\\Curso 2021-22\\TFG\\Archivos\\Fotos_TFG\\Fotos_camara"
observer.start()
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    observer.stop()
    observer.join()
```

Figura 22. Predecir.ipynb fragmento 3

3.4 Programa en C# para capturar imágenes

Una vez explicados los archivos donde creo la Red Neuronal Convolucional y hago predicciones con ella, voy a pasar a explicar el programa encargado de encender la cámara del ordenador, capturar una imagen y guardarla en el sistema de archivos para que el programa en Python encargado de hacer predicciones pueda coger esa imagen y gracias al entrenamiento de la CNN sea capaz de distinguir la persona que aparece en ella.

Este programa está escrito en el lenguaje de programación **C#** dentro del marco de trabajo de .NET porque ofrece mucha ayuda al programador para implementar sus propios programas de una manera muy sencilla y rápida.

Explicación del programa llamado **"Prueba_TFG.sln"**:

Antes de comenzar con las operaciones del propio programa tenemos un retardo de 30 s. para que la aplicación funcione correctamente.

```
private void Form1_Load(object sender, EventArgs e)
{
    Thread.Sleep(30000);
    EncenderCamara();

}
```

Figura 23. Prueba_TFG.sln fragmento 1

Una vez pasado ese tiempo, llamo al método **"EncenderCamara**" que se encarga de encender la cámara del ordenador y dibujar en pantalla la imagen que captura ésta gracias al método **"DrawLatestImage"**.

```
private void EncenderCamara()
{
    foreach (Camera cam in CameraService.AvailableCameras)
    {
        SetFrameSource(new CameraFrameSource(cam));

    }

    try
    {
        _frameSource.Camera.CaptureWidth = 320;
        _frameSource.Camera.CaptureHeight = 240;
        _frameSource.Camera.Fps = 20;
        _frameSource.NewFrame += OnImageCaptured;
        pictureBox1.Paint += new PaintEventHandler(DrawLatestImage);
        _frameSource.StartFrameCapture();

    }

    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());

    }
}
```

Figura 24. Prueba_TFG.sln fragmento 2

Seguidamente, guardo esa imagen en el sistema de archivos de mi ordenador, apago la cámara, arranco un temporizador de 5 min. para volver a realizar todo el proceso de nuevo cada 5 min. mientras el programa esté funcionando y también creo un manejador de eventos utilizando la clase **"FileSystemWatcher"** para que monitorice el directorio donde se encuentra el fichero **"Respuesta_CNN.txt"** y realice ciertas acciones cuando el archivo **"Predecir.ipynb"** modifique su contenido.

```
private void HacerFoto()
{
    if (_latestFrame != null)
    {
        Image imagen = _latestFrame;
        imagen.Save(@"C:\Users\Usuario\Desktop\Curso 2021-22\TFG\Archivos\Fotos_TFG\Fotos_camara\" + "Foto.jpg", ImageFormat.Jpeg);

        _frameSource.StopFrameCapture();

        timer1.Start();

        watcher.Path = @"C:\Users\Usuario\Desktop\Curso 2021-22\TFG\Archivos\CNN\";
        watcher.Filter = "Respuesta_CNN.txt";
        watcher.IncludeSubdirectories = false;
        watcher.Changed += new FileSystemEventHandler(OnChanged);
        watcher.EnableRaisingEvents = true;

    }
}
```

Figura 25. Prueba_TFG.sln fragmento 3

Por último, cuando el archivo **"Respuesta_CNN.txt"** es modificado llamo al método **"OnChanged"** creado por el manejador de eventos el cual se encarga de leer su contenido y guardarlo en una variable. Si esa variable contiene un **"1"** significa que aparezco yo en la imagen y si contiene un **"0"** significa que aparece mi madre. Si aparezco yo y es la primera vez, se muestra por pantalla un mensaje emergente diciendo **"Bienvenido Juan Guillermo** [(1)]**"** y si han pasado 30 min desde la primera foto en la que aparecí por primera vez, sale otro mensaje diciendo **"Llevas 30 minutos trabajando, te aconsejamos que te tomes un descanso** [(2)]**"**. Por el contrario, si aparece mi madre y es su primera vez, sale otro nuevo mensaje diciendo **"Usted no tiene permiso para utilizar este ordenador**[(3)]**"**.

```
private void OnChanged(object sender, FileSystemEventArgs e)
{
    WaitReady(@"C:\Users\Usuario\Desktop\Curso 2021-22\TFG\Archivos\CNN\Respuesta_CNN.txt");

    FileStream fs = new FileStream(@"C:\Users\Usuario\Desktop\Curso 2021-22\TFG\Archivos\CNN\Respuesta_CNN.txt", FileMode.Open, FileAccess.Read);
    StreamReader sr = new StreamReader(fs);
    respuesta = sr.ReadLine();
    sr.Close();
    fs.Close();

    if (respuesta == "1")
    {
        contador++;
        contador2 = 0;

        if (contador == 1)
        {
            MessageBox.Show("Bienvenido Juan Guillermo");

        }

        if (contador == 264)
        {
            MessageBox.Show("Llevas 30 minutos trabajando, te aconsejamos que te tomes un descanso");

        }

    }

    if (respuesta == "0")
    {
        contador = 0;
        contador2++;

        if (contador2 == 1)
        {
            MessageBox.Show("Usted no tiene permiso para utilizar este ordenador");

        }
```

Figura 26. Prueba_TFG.sln fragmento 4

La función del temporizador de 5 min. es la siguiente: A los 5 min. de hacer una imagen se repite todo el proceso; es decir, se enciende de nuevo la cámara y se representa la imagen en pantalla, se guarda esa imagen, se apaga ésta y esperamos a que el archivo **"Predecir.ipynb"** modifique el fichero de texto para que este programa sepa que persona aparece en la foto y saque el mensaje emergente correspondiente.

3.5 Funcionamiento de la aplicación

Para que la aplicación se ejecute automáticamente al encender el ordenador sin que un usuario la inicie manualmente es necesario crear los ejecutables de los programas "Predecir.ipynb" y "Prueba_TFG.sln" que reciben el nombre de: **"Predecir.exe"** y **"Prueba_TFG.appref-ms"** respectivamente.

Una vez tenemos los ejecutables creados, el funcionamiento de la aplicación es el siguiente: En primer lugar, se inicia **"Predecir.exe"** y 30 s. después **"Prueba_TFG.appref-ms"** captura la primera imagen para darle tiempo al otro ejecutable a estar preparado para recibirla. Esto solo ocurre con la primera imagen que capturamos, para las siguientes simplemente se captura la nueva imagen, ya que **"Predecir.exe"** estaría ya iniciado.

Seguidamente, cuando **"Predecir.exe"** recibe la imagen, ejecuta la función "predict" que muestra en pantalla el nombre de la persona que identifica en la imagen y escribe en el fichero "Respuesta_CNN.txt" un "1" o un "0" dependiendo de la persona que identifica.

Por último, **"Prueba_TFG.appref-ms"** lee ese fichero y según su lectura saca por pantalla el mensaje emergente correspondiente a la persona que aparece en esa imagen.

Cada 5 min. se captura una nueva foto y se identifica a la persona que aparece en ella siguiendo el mismo proceso descrito anteriormente de forma continuada mientras que el usuario no cierre manualmente ninguno de los dos ejecutables.

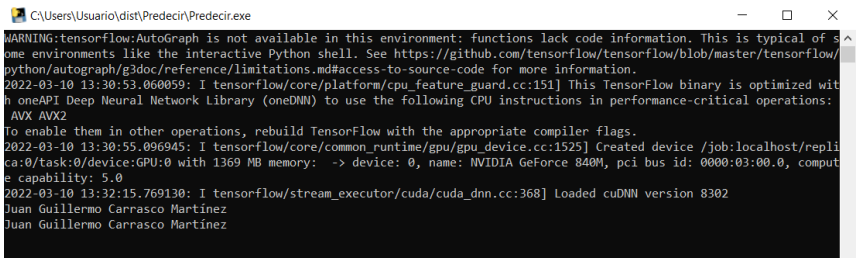

Figura 27. Predecir.exe

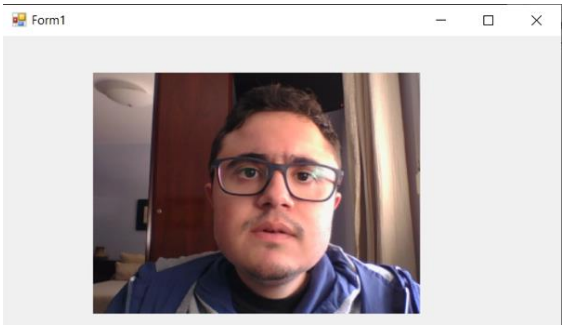

Figura 28. Prueba_TFG.appref-ms

Capítulo 4. Pruebas y resultados

Para realizar las pruebas de funcionamiento y comprobar que todo se desarrolla correctamente, he puesto en marcha mi aplicación para que capture fotos tanto mías como de mi madre y los resultados que he obtenido han sido los siguientes (Figuras 29-31).

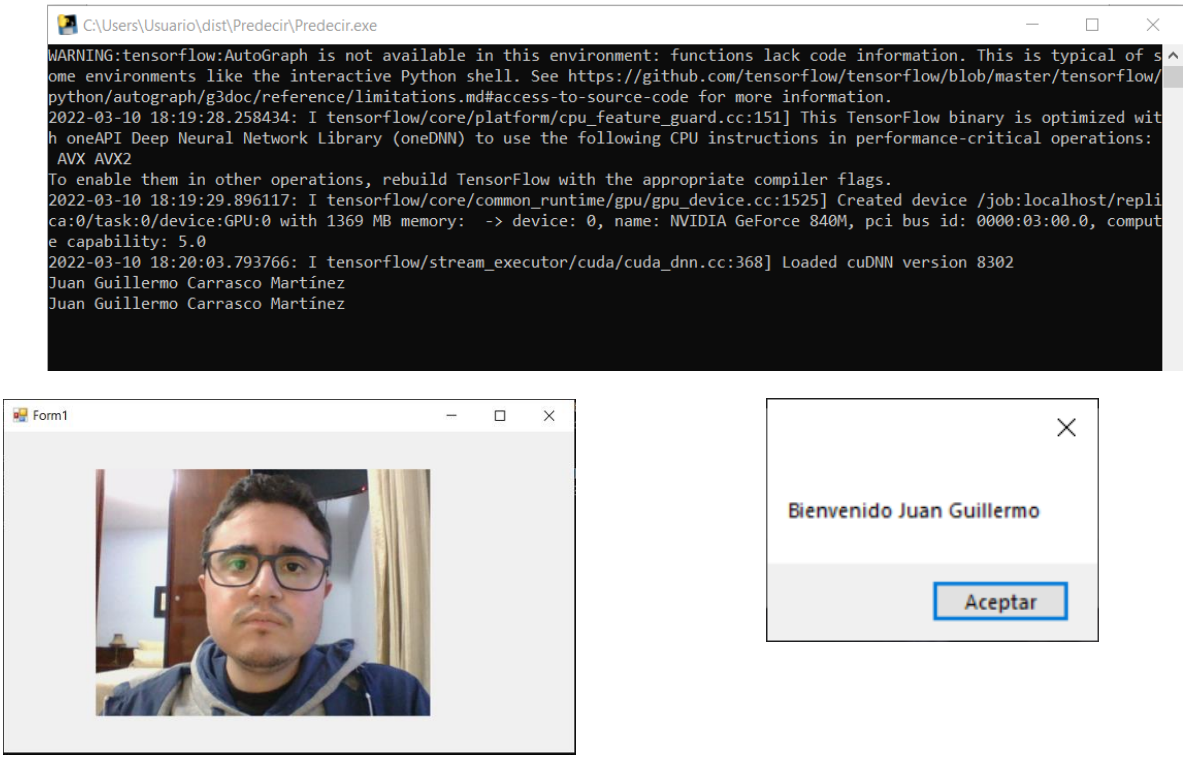

Figura 29. Prueba del mensaje (1)

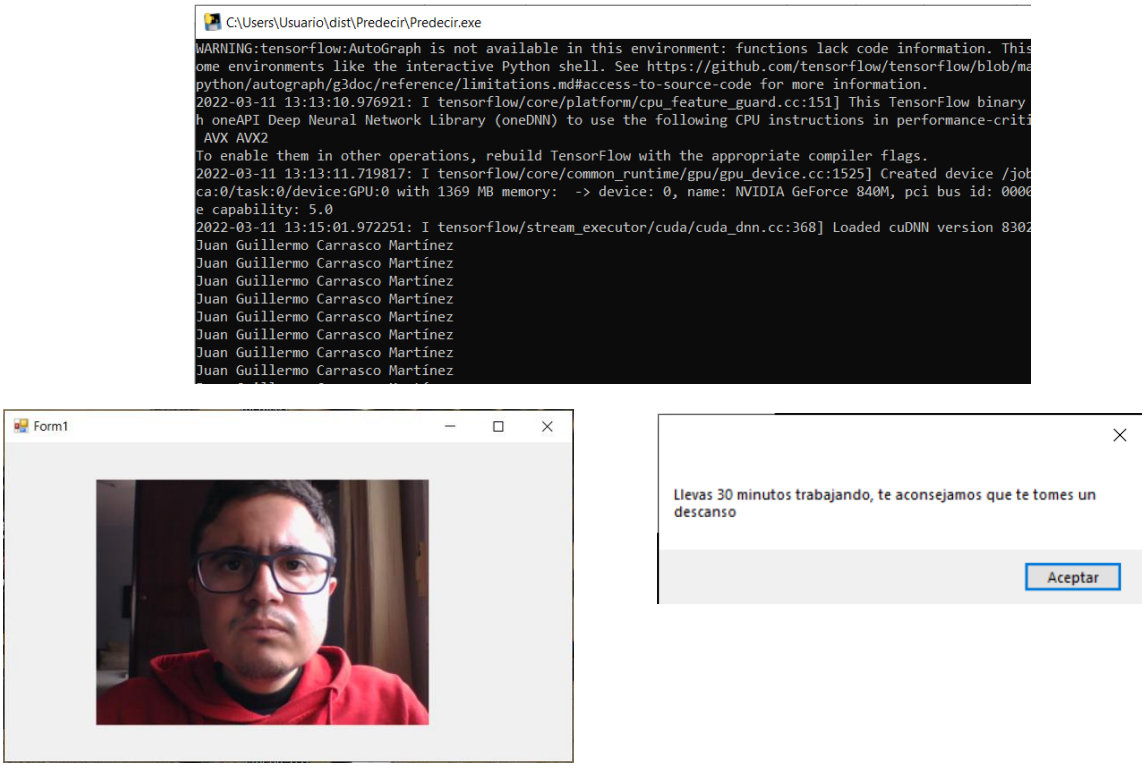

Figura 30. Prueba del mensaje (2)

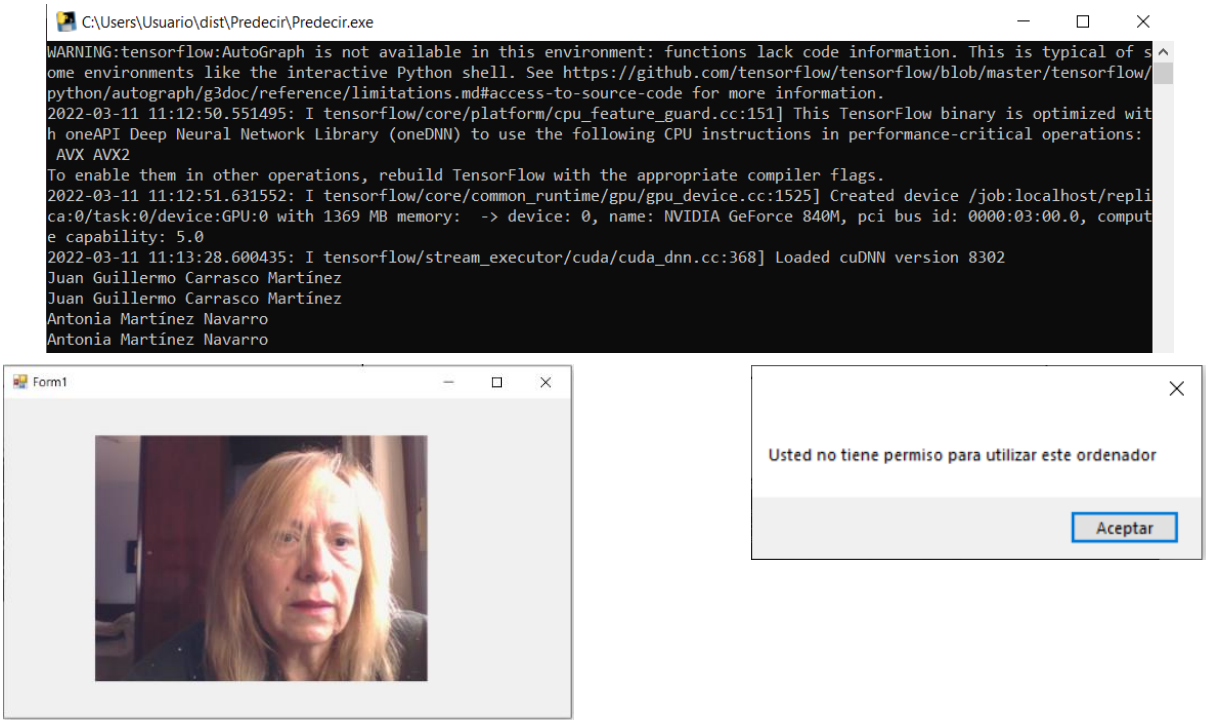

Figura 31. Prueba del mensaje (3)

Como podemos observar en las imágenes, los resultados que he obtenido han sido correctos, por lo que podemos concluir que la aplicación funciona sin ningún problema.

Capítulo 5. Conclusiones

5.1 Conocimientos aprendidos

Una vez finalizado este proyecto, he conseguido aprender las siguientes nociones que antes desconocía:

- Aprendizaje de la librería Matplotlib de Python para generar gráficas.
- Aprender a abrir un fichero de texto, escribir en él y cerrarlo en Python.
- Uso de la librería watchdog de Python para monitorizar y responder a eventos del sistema de archivos de un ordenador.
- Aprender a utilizar temporizadores en el lenguaje de programación C#.
- Aprendizaje de la clase FileSystemWatcher de .Net para la generación de eventos cuando cambia un directorio o un archivo de un directorio en un ordenador.
- Abrir un fichero de texto y leerlo mediante el lenguaje de programación C#.
- Saber crear ejecutables de Python y C#.
- Como introducir nuevas aplicaciones en el arranque de inicio de Windows.

5.2 Usos futuros

En un futuro, esta aplicación podría tener diversos usos muy interesantes, como, por ejemplo:

- Que al identificar a la persona que aparece en la imagen, ejecutara automáticamente programas que ese usuario suela utilizar con frecuencia como podría ser Spotify o Microsoft Word.
- Que al identificar en la imagen a una persona que no tenga permisos para iniciar sesión en el ordenador, éste se bloquee.
- Que cada cierto tiempo actualice el dataset de imágenes automáticamente para incluir más fotos y ser más preciso a la hora de identificar personas.
- Que compruebe el tiempo que pasa cada usuario (en caso de tener varios) delante del ordenador cada día y al cabo de una semana te presente un informe del tiempo de uso de la semana anterior.
- Que cuando lleves demasiado tiempo al día delante del ordenador, te aparezca un mensaje en pantalla y se te apague el ordenador durante 30 min. para que hagas un descanso.

5.3 Cierre

Este proyecto me ha permitido aprender un poco más acerca de la Inteligencia Artificial, concretamente sobre el Deep Learning y el aprendizaje supervisado. Esta tecnología nos abre nuevos caminos por explorar que nos pueden facilitar la vida cotidiana en muchos aspectos, pero por el contrario para que funcionen correctamente necesitamos obtener muchos datos de entrenamiento, acción que en algunos casos puede llegar a ser muy complicada.

Actualmente, este campo está en constante evolución y desarrollo, lo que quiere decir que en los próximos años estará presente en muchas acciones que realizamos todos los días, como por ejemplo abrir la puerta de tu casa automáticamente al llegar.

Cada vez más, los ordenadores y sistemas que existen en el mercado tienen mejores prestaciones lo que significa que una persona con ciertos conocimientos que disponga de un ordenador y conexión a Internet es capaz de experimentar y usar esta tecnología tan interesante. De esta manera, muchas personas van a poder desarrollarse sus propias aplicaciones que utilizan IA para funcionar.

En definitiva, este TFG me ha servido para tener mi primera experiencia con la IA, descubrir sus innumerables usos y formarme en un campo que sin duda va a ser predominante en un futuro no muy lejano y va a demandar de muchos profesionales.

Referencias

[1] "Inteligencia Artificial, Machine Learning, Deep Learning," *https://www.yeeply.com/blog/inteligencia-artificial-machine-deep-learning/*

[2] "Ilustración conceptos teóricos," *https://www.devacademy.es/machine-learning-deep-learning-e-inteligencia-artificial-mundo-habla-ello*

[3] "Algoritmos de Machine Learning," *https://www.auraquantic.com/es/tipos-de-algoritmos-de-inteligencia-artificial-y-machine-learning/*

[4] "Estructura de una CNN," *https://towardsml.wordpress.com/2018/10/16/deep-learning-series-p2-understanding-convolutional-neural-networks/*

[5] "Ilustración Convolución," *https://ichi.pro/es/kernels-filters-en-la-red-neuronal-convolucional-cnn-hablemos-de-ellos-16376329614105*

[6] "Ilustración Max-Pooling," *https://es.quora.com/Qu%C3%A9-es-Max-Pooling-en-CNN*

[7] "Ilustración capa totalmente conectada," *https://datapeaker.com/big-data/arquitectura-de-red-neuronal-convolucional-arquitectura-cnn/*

[8] "Redes Neuronales Convolucionales (CNN)," *https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/*