

Universidad Internacional de La Rioja (UNIR)

Escuela de Ingeniería

Máster en Análisis y Visualización de Datos Masivos

Herramienta de reconocimiento facial y emociones para soporte a personas ciegas

Trabajo Fin de Máster

presentado por: Vicente Gómez, Jesús María

Director/a: González Crespo, Rubén

Ciudad: Dublin

Fecha: 22/06/2017

RESUMEN

El presente proyecto describe el desarrollo de una herramienta de software para ser utilizada por personas ciegas y a las cuales les facilite obtener información en forma auditiva a partir de información presentada en formato visual. El análisis de la imagen de entrada se centra en detectar posibles caras y clasificar el estado emocional, así como el género de cada una de ellas. La herramienta a su vez permite guardar en el sistema la identidad de la persona detectada en la imagen para que posteriormente sea utilizado durante la etapa de análisis y esta identidad sea transmitida también en formato auditivo al usuario. La herramienta da soporte en varios idiomas y consta de un menú sonoro y de captura de comandos de voz para facilitar la interacción con el mismo. Para el desarrollo del sistema se han utilizado técnicas de inteligencia artificial como las redes neuronales, así como herramientas que permiten la conversión de texto a voz y voz a texto.

Palabras Clave: Reconocimiento facial, Reconocimiento de emociones, Texto a voz, Voz a texto, Accesibilidad

ABSTRACT

Current project describes the development of software tool to be used by blind people to help them understand visual information by presenting it in audio format. An input image is analysed to detect any existing faces in it and for each face classify de gender and emotion of the person. The tool also allows to store in the system the identity of the person detected so it can be used later in the analysis stage and output this information also in audio format. Multiple languages are supported and to ease the interaction with the user the system as an audio menu and captures spoken commands. On the development of the tool AI techniques including neural networks and text to voice and voice to text tools have been used.

Keywords: Facial recognition, Emotion recognition, Text to Voice, Voice to Text, Accessibility

INDICE DE CONTENIDO

RESUMEN	2
ABSTRACT	3
INDICE DE CONTENIDO	4
INDICE DE ILUSTRACIONES	7
INDICE DE TABLAS	9
CAPÍTULO 1	10
INTRODUCCIÓN	10
1.1 Motivación	10
1.2 Planteamiento del trabajo	11
1.3 Estructura del trabajo:	12
CAPÍTULO 2	14
CONTEXTO Y ESTADO DEL ARTE	14
2.1 Contexto y estado del arte	14
2.1.1 Evolución cronológica de la IA	14
2.1.2 Métodos de la IA	15
2.1.3 Redes Neuronales	17
2.1.4 Redes Neuronales Convolucionales (CNNs)	19
2.1.5 Técnicas de segmentación de las CNNs	24
2.1.6 Herramientas comerciales para el reconocimiento facial	28
2.2 Conclusiones	30
CAPÍTULO 3	31
METODOLOGIA DE TRABAJO Y OBJETIVOS CONCRETOS	31
3.1 Objetivo general	31
3.2 Objetivos específicos	31
3.3 Metodología de trabajo	31
CAPÍTULO 4	33
DESARROLLO ESPECÍFICO DE LA CONTRIBUCIÓN	33
4.1 Identificación de requisitos	33
4.1.1 Requisitos funcionales	33
4.1.2 Requisitos no funcionales	34
4.2 Descripción detallada del producto resultante	34

4.2.1	Entorno de desarrollo	34
4.2.2	Modelos clasificatorios	39
4.2.2.1	Sistema de detección de caras.....	40
4.2.2.2	Sistema de reconocimiento de caras	44
4.2.2.3	Sistema de clasificación de emociones	46
4.2.2.4	Sistema de clasificación de género	51
4.2.3	Sistema de conversión voz a texto.....	52
4.2.4	Sistema de conversión texto a voz.....	54
4.2.5	Integración de los modelos	56
4.2.5.1	Inicialización	58
4.2.5.2	Detección de caras	59
4.2.5.3	Procesamiento de caras	61
4.2.5.4	Opciones del sistema	66
4.2.5.4.1	Audio de las opciones	66
4.2.5.4.2	Diagrama de flujo del menú de opciones	69
4.2.5.4.3	Opción LANGUAGE.....	72
4.2.5.4.4	Opción WHO.....	72
4.2.5.4.5	Opción SAVE.....	75
4.2.5.4.6	Opción CANCEL	78
4.2.5.4.7	Opción QUIT	78
4.2.6	Sistema de soporte multi-lenguaje	78
4.3	Evaluación de la calidad y aplicabilidad del producto.....	87
CAPÍTULO 5	88
CONCLUSIONES Y TRABAJOS FUTUROS	88
5.1	Resumen del problema	88
5.2	Resumen de las contribuciones	88
5.3	Líneas de trabajo futuro.....	91
CAPÍTULO 6	93
REFERENCIAS Y ENLACES.....	93
CAPÍTULO 7	98
ANEXOS	98
7.1	Instrucciones de instalación.....	98
7.2	Instrucciones para habilitar la interacción por voz	98
7.3	Paquetes Python	99

7.4 Script de generación de archivos de audio utilizados	104
--	-----

INDICE DE ILUSTRACIONES

Figura 1: Proceso de reconocimiento de patrones (Elisfm, 2009).....	16
Figura 2: Tendencia en las búsquedas de los diferentes métodos (Google, 2017).....	17
Figura 3: Red neuronal simple (Wikipedia, 2017)	17
Figura 4: Perceptrón multicapa (Karpathy, CS231n Convolutional Neural Networks for Visual Recognition, 2016)	18
Figura 5: Ejemplos de descripción de imágenes utilizando redes neuronales (Karpathy, http://cs.stanford.edu/ , 2017).....	19
Figura 6: Ejemplos de reconocimiento de objetos utilizando redes neuronales (Shaoqing Ren, 2016).....	20
Figura 7: Arquitectura de red neuronal convolucional (Karn, An Intuitive Explanation of Convolutional Neural Networks, 2016)	21
Figura 8: Representación matricial de una imagen de entrada (Geitgey, 2016).....	21
Figura 9: Convolución (Convolution Matrix, 2014)	22
Figura 10: Imagen de ejemplo (Kernel image processing, 2017)	22
Figura 11: Ejemplo de sub-muestreo (Karpathy, CS231n Convolutional Neural Networks for Visual Recognition, 2016).....	24
Figura 12: Evolución de la tasa de error ImageNet (Parthasarathy, 2017).....	24
Figura 13: R-CNN	25
Figura 14: Clasificación de objetos (Malik, 2014)	26
Figura 15: Fast R-CNN (Ross Girshick, 2015).....	26
Figura 16: Faster R-CNN (Shaoqing Ren, 2016)	27
Figura 17: Clasificación de objetos con Faster R-CNN (Shaoqing Ren, 2016)	27
Figura 18: Mask R-CNN (Kaiming He, 2017).....	28
Figura 19: Segmentación por pixel (Kaiming He, 2017).....	28
Figura 20: Logo de Python (Python, 2017)	34
Figura 21: Anaconda (Anaconda, 2017)	35
Figura 22: Jupyter Notebook en funcionamiento	37
Figura 23: Ejemplo de archivo ipynb	38
Figura 24: Características Haar (Open CV, 2017)	41
Figura 25: Ejemplo Haar (Open CV, 2017)	41
Figura 26: Ejemplo archivo xml generado	42
Figura 27: Resultado de la detección (Open CV, 2017).....	43
Figura 28: Ejemplo de reconocimiento de rasgos faciales	45
Figura 29: Ejemplo de reconocimiento de identidades	45

Figura 30: Origen de los datos utilizados (Rasmus Rothe, 2015)	52
Figura 31: Diagrama de flujo de la herramienta.....	57
Figura 32: Diagrama de flujo de la inicialización	58
Figura 33: Diagrama de flujo de la detección de caras	60
Figura 34: Diagrama de flujo del procesamiento de caras	62
Figura 35: Diagrama de flujo del menú de opciones.....	69
Figura 36: Diagrama de flujo de la opción guardar	77
Figura 37: Audios de identidades guardadas en el sistema.....	78
Figura 38: Imágenes de identidades guardadas en el sistema	78
Figura 39: Ejemplo multi-lenguaje	80

INDICE DE TABLAS

Tabla 1: Ejemplos de funciones de activación (Karn, A Quick Introduction to Neural Networks, 2016).....	18
Tabla 2: Ejemplo de diferentes filtros aplicados a una imagen (Kernel image processing, 2017).....	23
Tabla 3: Comparativa de soluciones comerciales (Kairos, 2017).....	29
Tabla 4: Hitos del proyecto.....	32
Tabla 5: Requisitos funcionales.....	34
Tabla 6: Requisitos no funcionales.....	34
Tabla 7: Resultados del análisis de la imagen.....	63
Tabla 8: Comandos de entrada.....	66
Tabla 9: Audios de la interacción.....	68
Tabla 10: Audios del análisis.....	69
Tabla 11: Descripción de las opciones.....	70
Tabla 12: Ejemplos de análisis.....	75
Tabla 13: Archivos de idioma por género.....	82
Tabla 14: Requisitos y contribuciones.....	91

CAPÍTULO 1

INTRODUCCIÓN

1.1 Motivación

Tradicionalmente la forma de interactuar con las máquinas se ha realizado de una manera explícita: programadores o expertos crean programas que se basan en una serie de instrucciones que la máquina es capaz de entender para que posteriormente sea utilizada por una serie de usuarios concretos. Éstos usuarios más tarde interactúan con la máquina de forma explícita al introducir datos o efectuar alguna acción de las disponibles en programa o sistema.

Los programas normalmente no han sido programados para ir más allá de esta interacción explícita entre usuario y máquina. La máquina solo actúa en consecuencia a lo que el usuario decide.

Por otra parte, la limitación visual de las personas ciegas hace que la información que como usuarios reciben a través de un programa o aplicación esté restringido. Por ejemplo, en el ámbito concreto de identificar lo que una imagen o fotografía representa para obtener información a través de ella o en el caso de ser capaz de identificar el estado de ánimo de un interlocutor a partir de sus gestos fáciles.

Esta limitación se da también en el sentido inverso al dificultarse la comunicación desde el usuario a la máquina a la hora de interactuar con la información recibida.

Hasta hace bien poco estas dificultades eran extensibles para las máquinas las cuales han sido capaces de realizar tareas complejas para los humanos de forma rápida y sencilla pero luego eran incapaces de realizar tareas que para los humanos son por lo general innatas y fáciles como ser capaz de analizar los elementos de una imagen o entender instrucciones que no se encontraran dentro de la lista predefinida y en un formato determinado por los programadores.

Con los avances en los últimos años en inteligencia artificial debidos en gran medida al incremento en la capacidad de procesamiento y al gran volumen de información que constantemente se genera este problema ha disminuido notablemente habiendo casos incluso en los que las máquinas superen a los humanos si bien todavía queda camino que recorrer e incluso mejorar no ya centrándonos en los resultados sino en la eficiencia y la versatilidad de las técnicas utilizadas.

Es por estos avances que considero que se puede intentar abordar los problemas de interacción de las personas ciegas con un programa o un interlocutor haciendo uso de técnicas de inteligencia artificial principalmente y centrándome en mejorar la información transmitida al usuario concretamente en el ámbito de la interacción con un interlocutor.

Concretamente, quiero tratar los siguientes problemas a los que se enfrentan las personas ciegas:

- La dificultad para ser capaces de identificar en una imagen de entrada (ya sea en un archivo de imagen, *streaming* de video, videollamada o incluso en persona) la existencia de una persona o personas.
- La dificultad para ser capaces de identificar características determinantes de dicha persona o personas como por ejemplo el sexo o la edad.
- La dificultad para ser capaces de identificar características circunstanciales concretas de dicha persona o personas como por ejemplo el estado de ánimo.
- La dificultad para ser capaces de identificar como personas conocidas a dichas persona o personas.
- La dificultad para ser capaces de interactuar con un programa de forma no visual.

1.2 Planteamiento del trabajo

Para solucionar dichas dificultades considero que hay dos partes importantes a la hora de plantear el trabajo:

- Primero: solventar los problemas del usuario para que a través de información que se presente visualmente (concretamente una imagen) y que debido a la situación del usuario no pueda ser entendible facilitar su comprensión.

Para ello se propone hacer uso de técnicas de inteligencia artificial para clasificación y reconocimiento de imágenes. Concretamente a partir de una imagen de entrada se planea:

- Identificar la persona o personas presentes.
- Identificar el sexo de dichas personas.
- Identificar el estado de ánimo de dichas personas.

- Identificar si las personas son conocidas o no.
- Segundo: facilitar en la medida de lo posible la interacción con el usuario teniendo en cuenta que la interacción visual es un impedimento. Para ello se propone hacer uso de las siguientes técnicas:
 - **Text To Speech:** para transmitir información desde el sistema hacia el usuario. Concretamente se planea hacer uso de esta técnica para transmitir al usuario los resultados obtenidos al enfrentarse al problema anterior.
 - **Speech To Text:** para recibir información del usuario hacia el sistema. Es decir, facilitar una serie de comandos auditivos para que el usuario pueda interactuar con el sistema.

1.3 Estructura del trabajo:

A continuación, paso a detallar brevemente los diferentes capítulos en los que está estructurada esta memoria:

- **Capítulo 1. Introducción:** En este capítulo se identifica el problema y se plantea cómo se va a solucionar. Además, se describe brevemente lo que se va a contar en cada uno de los capítulos de la memoria.
- **Capítulo 2. Contexto y estado del arte:** En este capítulo se describe el contexto de aplicación aportando un resumen del conocimiento ya existente en el campo del reconocimiento y clasificación de imágenes, así como de otras técnicas utilizadas para la solución del problema planteado. Se hace un recorrido a través de la historia de la inteligencia artificial para después ir centrándose en los métodos que se van a utilizar para el desarrollo de la herramienta, concretamente redes neuronales convolucionales.
- **Capítulo 3. Metodología de trabajo y objetivos concretos:** En es este capítulo se concretan los objetivos del proyecto y se describen la serie de pasos o hitos que se han marcado para la consecución del objetivo.
- **Capítulo 4: Desarrollo específico de la contribución:** En este capítulo se comienza identificando los requisitos del proyecto para luego pasar a relatar en detalle todo el proceso de desarrollo de la aplicación dividido en secciones para cada

elemento. Se incluye diagramas de flujo y código de ejemplo para las funciones principales.

- **Capítulo 5: Conclusión y trabajos futuros:** En este capítulo se hace un resumen del problema y de la solución planteada comparando los requisitos identificados primeramente con las soluciones planteadas para ellos. Además, se relatan posibles mejores y ampliaciones del sistema.
- **Capítulo 6: Referencias y enlaces:** Listado de referencias y bibliografía utilizados en el proyecto.
- **Capítulo 7: Anexo:** Elementos no incluidos en la memoria inicialmente.

CAPÍTULO 2

CONTEXTO Y ESTADO DEL ARTE

2.1 Contexto y estado del arte

El estudio y desarrollo de la inteligencia artificial es un campo que en los últimos años está en expansión debido al reciente avances en la capacidad de procesamiento y en las grandes cantidades de datos que se almacenan actualmente incrementándose día a día. Teóricamente muchos de los logros conseguidos actualmente se basan en conceptos e ideas no nuevos, pero sólo recientemente debido a los avances anteriormente mencionados se están consiguiendo cosas que antes eran impensables.

En este capítulo voy a empezar poniendo en contexto la evolución a nivel histórico de la inteligencia artificial para después enumerar y describir los tres principales métodos o aproximaciones que se han realizado. A continuación, pasaré a explicar las redes neuronales artificiales que es un modelo computacional inspirado en las neuronas biológicas para después centrarme en un tipo concreto de estas: las redes neuronales convolucionales. Posteriormente me centraré en diversas mejoras aplicada a la clasificación de objetos dentro de este tipo redes y para finalizar mostraré una comparativa de diferentes herramientas comerciales para el ámbito concreto de clasificación facial en imágenes.

Pasó a continuación a poner en contexto brevemente la evolución en los avances en inteligencia artificial:

2.1.1 Evolución cronológica de la IA

- **Antes de 1952:** desde la antigüedad se muestra interés en la búsqueda de un modo resolutor capaz de ganar con los mínimos movimientos posibles (Torres de Hanoi, hacia el 3000 a.C.).

Distintas civilizaciones a lo largo de la historia han construido autómatas siendo los más antiguos las estatuas sagradas del antiguo Egipto.

En el 300 a.C. Aristóteles describe un conjunto de reglas de forma estructurada para describir el funcionamiento de la mente humana.

En 1315 Ramon Llull tuvo la idea de que el razonamiento podía ser efectuado de manera artificial.

En 1937 Alan Turing publica un artículo que establece las bases teóricas para todas las ciencias de computación, en 1940 construye el primer computador

electromecánico y en 1041 Konrad Zuse crea la primera computador programable y el primer lenguaje de programación de alto nivel.

- **1952 - 1956, Nacimiento de la inteligencia artificial:** Se crea el término “Inteligencia Artificial” durante una conferencia (Darmouth) y se empieza a discutir la posibilidad de crear un cerebro artificial.
 - **1956 - 1974, Los años de oro:** Fueron años de descubrimientos sobre un nuevo campo. Los programas creados durante este tiempo eran capaces de resolver problemas algebraicos, demostrar teoremas geométricos y aprender a hablar inglés.
 - **1974 - 1980, Primer invierno de la Inteligencia Artificial:** Se descubre que el tremendo optimismo ha elevado las expectativas demasiado y los investigadores de inteligencia artificial empiezan a darse cuenta de que no han sabido comprender la dificultad de los problemas a los que se enfrentaban.
 - **1980 - 1987, Boom de la Inteligencia Artificial:** Nacen los sistemas expertos que son que dan respuestas a problemas en un dominio específico.
 - **1987 - 1993, Segundo invierno de la Inteligencia Artificial:** Una vez más las expectativas superaron los resultados lo que hizo que la inversión en inteligencia artificial se redujera drásticamente.
 - **1993 - 2001, Nueva Inteligencia Artificial:** Los adelantos en capacidad de procesamiento hacen que antiguas metas de la inteligencia artificial sean alcanzadas.
 - **2000 - Presente, Deep Learning, Big Data e Inteligencia Artificial General:** Las grandes cantidades de información (Big Data), el incremento en la capacidad de procesamiento y nuevas técnicas de “machine learning” se aplican satisfactoriamente a gran variedad de problemas haciendo resurgir el interés en la inteligencia artificial.
- En la siguiente figura se puede observar la tendencia

2.1.2 Métodos de la IA

En este apartado se van a presentar las diferentes aproximaciones que con las que se ha intentado resolver el problema de la inteligencia artificial identificando tres puntos principales de vista:

- **Reconocimiento de patrones:**
El reconocimiento de patrones era un término muy popular en los años 70 y 80. Consiste de tres procesos que se pueden observar en la siguiente figura

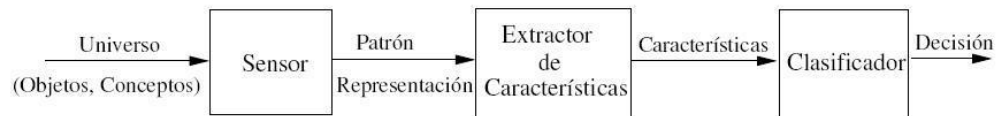


Figura 1: Proceso de reconocimiento de patrones (Elisfm, 2009)

Técnicas como los árboles de decisión o el análisis cuadrático discriminante se desarrollaron en este sistema.

- **Aprendizaje automático:**

En los años 90 se empezó a descubrir que una manera más eficaz de crear algoritmos de reconocimiento de patrones es reemplazar al experto en el dominio con gran cantidad de información, es decir, el paso de aprendizaje se realiza de manera explícita. Posteriormente, a mediados de 00 estas ideas se empezaron a utilizar no sólo en reconocimiento de patrones en imágenes sino a otros ámbitos como la robótica, los mercados financieros o el genoma humano con gran éxito. El problema de este sistema es que existen multitud de algoritmos disponibles para utilizar cada uno con sus diferentes parámetros que ajustar e ir probando iterativamente hasta encontrar la solución más óptima para un problema concreto.

- **“Deep Learning”:**

El enfoque de este sistema es enfatizar el tipo de modelo que se quiere utilizar (por ejemplo, una red neuronal convolucional multi capa) y en vez de tener que ajustar los parámetros iterativamente como en el enfoque anterior esto se reemplaza con mucha información. El problema que presenta es que se requiere mucha cantidad de información y mucha capacidad de procesamiento debido a la alta multidimensionalidad.

A continuación, se muestra una comparativa de popularidad entre los diferentes términos presentados:

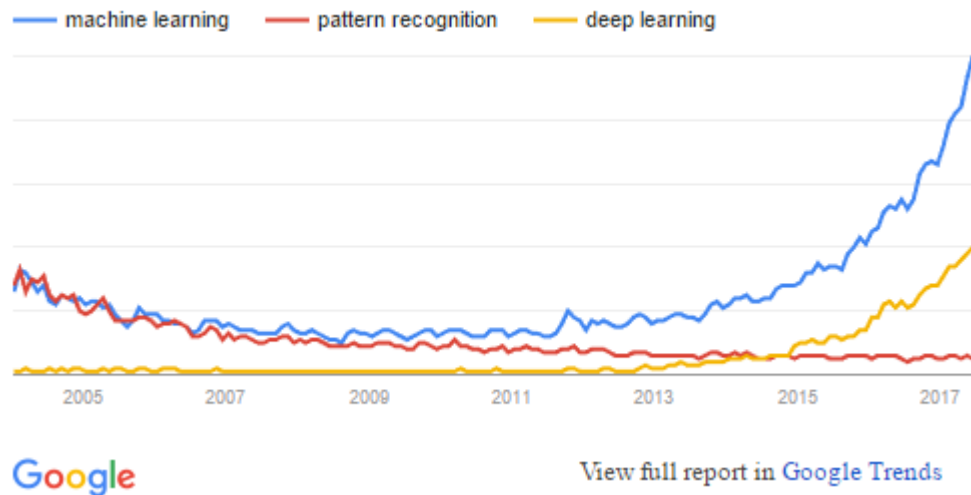


Figura 2: Tendencia en las búsquedas de los diferentes métodos (Google, 2017)

2.1.3 Redes Neuronales

Una red neuronal artificial es un modelo computacional inspirado en la manera de procesar la información de las redes neuronales biológicas.

La unidad básica de este sistema es la neurona que recibe unos valores de entrada de una fuente externa o de otras neuronas y genera un valor de salida. Cada valor de entrada tiene asignado un peso basado en la importancia de esa entrada. La neurona aplica una función de activación al valor calculado de la suma de todas las entradas multiplicadas por sus respectivos pesos para obtener el valor de salida.

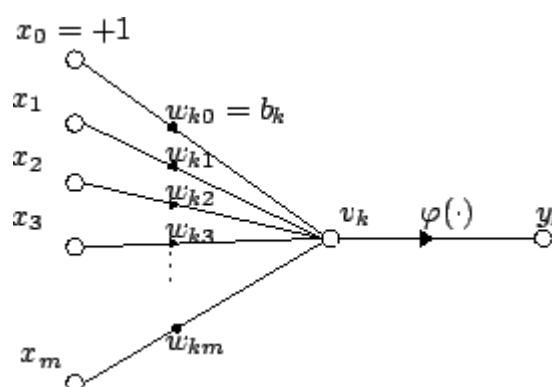


Figura 3: Red neuronal simple (Wikipedia, 2017)

El objetivo de la función de activación es introducir no linealidad y algunas de las más utilizadas son la función sigmoide, la tangente hiperbólica o la rectificación lineal (ReLU).

Sigmoide	Tangente hiperbólica	Rectificación lineal
----------	----------------------	----------------------

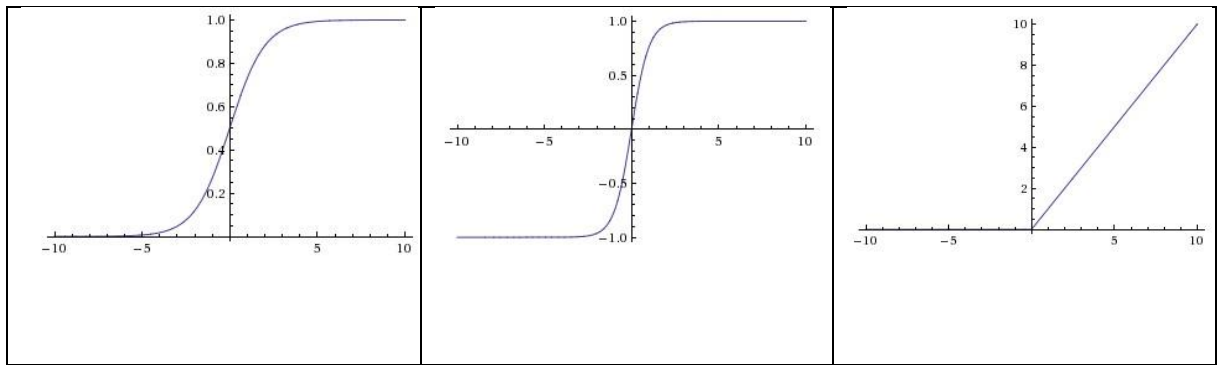


Tabla 1: Ejemplos de funciones de activación (Karn, *A Quick Introduction to Neural Networks*, 2016)

La red neuronal más básica y simple ideada fue la red neuronal prealimentada (o *feed forward* en inglés). Consiste en una serie de neuronas organizadas en capas y las neuronas de cada capa están conectadas con las neuronas de las capas adyacentes. Esto hace que las neuronas se puedan clasificar en tres tipos: de entrada, ocultas y de salida. (Karpathy, CS231n Convolutional Neural Networks for Visual Recognition, 2016).

En este tipo de red la información fluye en una única dirección: hacia adelante, es decir, de las neuronas de entrada a las de salida pasando por las ocultas.

Ejemplos de este tipo de red son:

- **Perceptrón de una capa:**
Es la más simple, no tiene capas ocultas.
- **Perceptrón multicapa**
Contiene por lo menos una capa oculta. En el siguiente ejemplo se muestra una red de este tipo con dos capas ocultas:

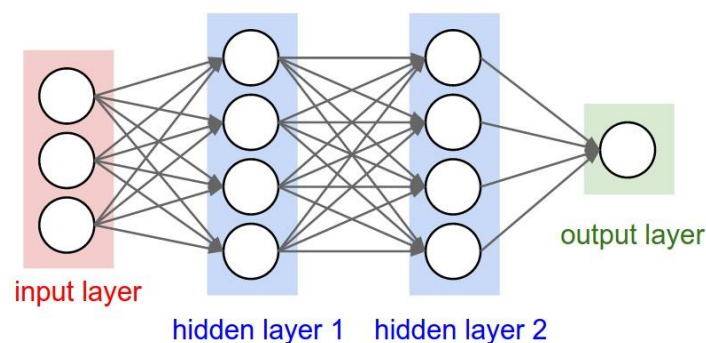


Figura 4: Perceptrón multicapa (Karpathy, CS231n Convolutional Neural Networks for Visual Recognition, 2016)

La forma en la que una red neuronal “aprende” es a través del algoritmo de propagación hacia atrás, siendo uno de los más populares el de propagación hacia atrás de los errores que consiste en aprender de los errores, es decir, se va corrigiendo. (Karn, A Quick Introduction to Neural Networks, 2016)

2.1.4 Redes Neuronales Convolucionales (CNNs)

Las redes neuronales convolucionales (CNNs a partir de ahora) son un tipo de redes neuronales que se han demostrado muy efectivas en reconocimiento y clasificación de imágenes, así como de procesamiento de lenguaje natural siendo esto un tipo de clasificación también sólo que en vez de imágenes sería clasificación de frases. A continuación, se muestra unos ejemplos de imágenes y la interpretación realizada de ellas utilizando CNNs para el reconocimiento de imágenes.

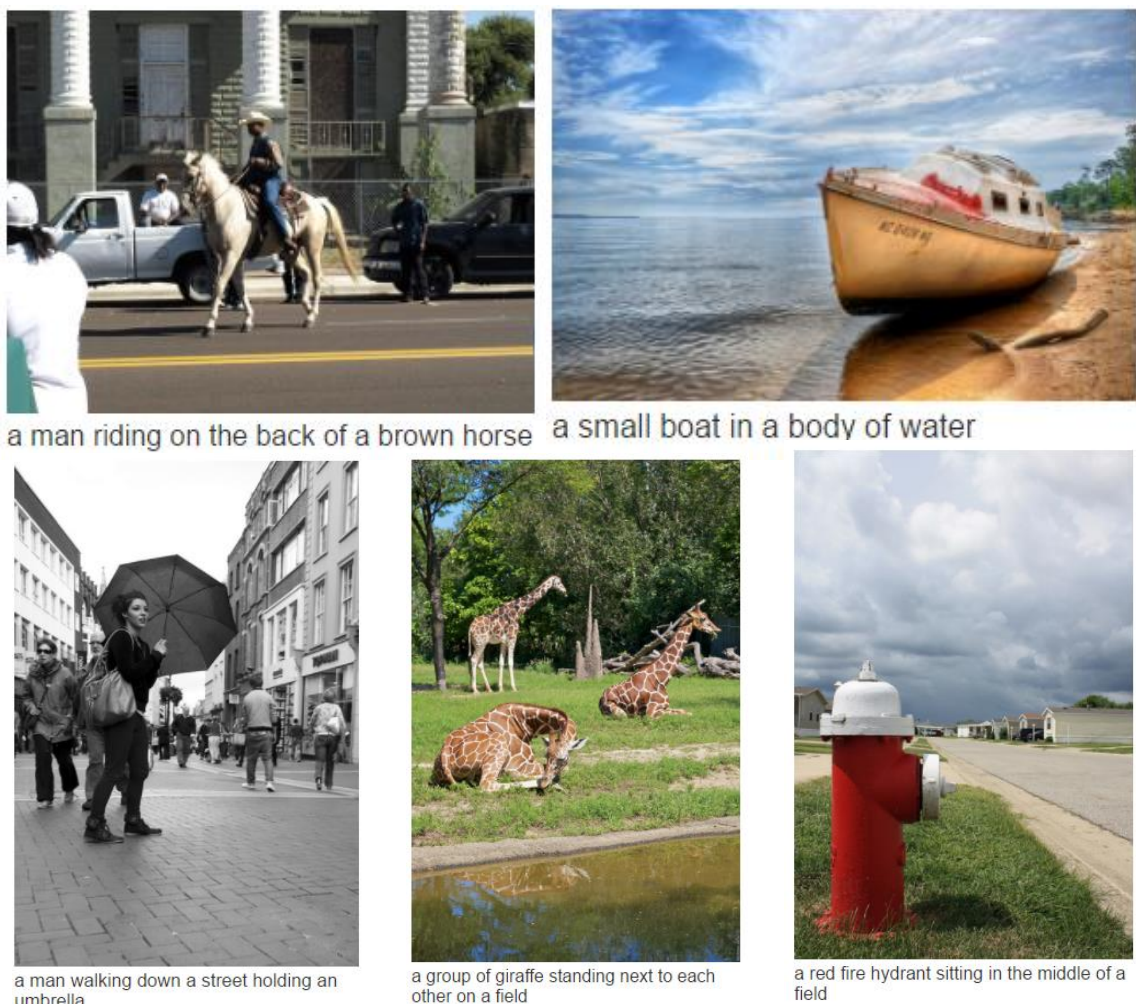


Figura 5: Ejemplos de descripción de imágenes utilizando redes neuronales (Karpathy, <http://cs.stanford.edu/>, 2017)

Como se ha comentado las CNNs también son muy efectivas a la hora de clasificar elementos de una imagen pudiendo ser estos objetos, personas, animales como se puede ver en los ejemplos siguientes:



Figura 6: Ejemplos de reconocimiento de objetos utilizando redes neuronales (Shaoqing Ren, 2016)

LeNet fue la primera red neuronal convolucional (Yan LeCun, 1998). Fue desarrollada en los años 90 por Yann LeCun de forma iterativa siendo su quinta versión (LeNet5) la más popular. Su tarea era reconocimiento de caracteres, concretamente caracteres numéricos como los de los códigos postales. Actualmente se están desarrollando constantemente nuevas arquitecturas, pero todas utilizan las ideas establecidas en LeNet siendo ésta utilizada para hacer entendibles los conceptos.

La siguiente imagen muestra un ejemplo de una CNN similar en arquitectura a LeNet y que clasifica una imagen de entrada en cuatro posibles categorías: perro, gato, barco o pájaro. En el ejemplo la entrada es la imagen de un barco y la red correctamente asigna una alta probabilidad a que sea un barco (0.94).

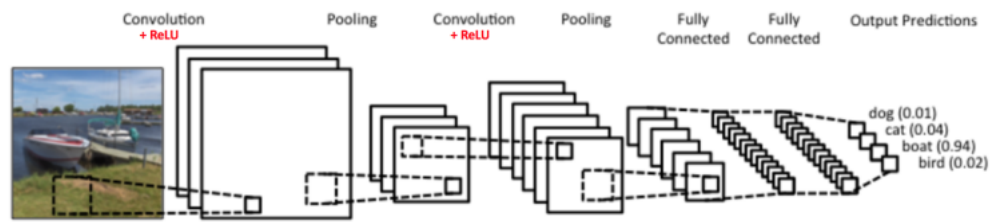


Figura 7: Arquitectura de red neuronal convolucional (Karn, An Intuitive Explanation of Convolutional Neural Networks, 2016)

La imagen de entrada es representada como una matriz de los valores que recibe cada pixel. Habitualmente las imágenes utilizan tres canales: rojo, verde y azul. Cada uno de estos canales se representa en una matriz de dos dimensiones cuyos elementos son los valores de cada pixel en un rango de 0 a 255. Las imágenes en blanco y negro utilizan un solo canal por lo que la imagen se puede representar como una sola matriz de dos dimensiones. (Geitgey, 2016)

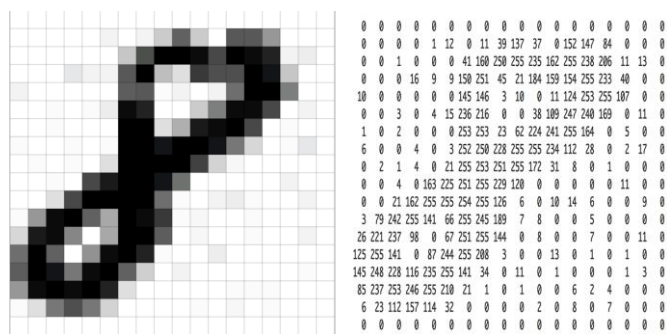


Figura 8: Representación matricial de una imagen de entrada (Geitgey, 2016)

Las operaciones realizadas en la CNN son las siguientes:

- **Convolución**

La tarea principal es extraer de la imagen características preservando la relación espacial entre píxeles aprendiendo características de la imagen utilizando pequeños rectángulos de los datos de entrada. Utilizando como ejemplo para ilustrar una imagen de 5x5 píxeles y un solo canal como la mostrada debajo y otra matriz de 3x3 (normalmente denominada filtro o kernel), al aplicar la convolución sobre los elementos englobados por el recuadro verde de la matriz original se obtiene lo siguiente:

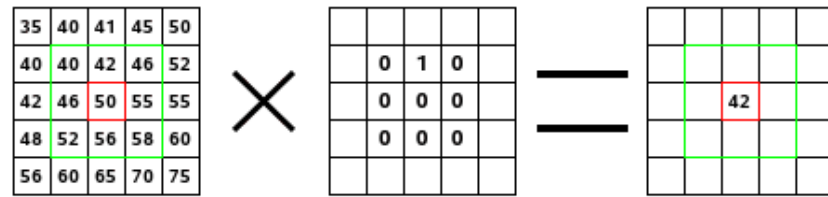


Figura 9: Convolución (Convolution Matrix, 2014)

A este resultado se le denomina mapa de activación o mapa de características. Queda claro que utilizando diferentes filtros se obtendrán mapas de activaciones diferentes, por ejemplo, partiendo de la siguiente imagen:



Figura 10: Imagen de ejemplo (Kernel image processing, 2017)

Se puede observar en la siguiente tabla las imágenes resultantes al aplicar diferentes filtros:

Operación	Filtro	Resultado
Identidad	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Detección de bordes I	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Detección de bordes II	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	

Detección de bordes III	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Refinamiento	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Emborronamiento	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Emborronamiento Gaussiano	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Tabla 2: Ejemplo de diferentes filtros aplicados a una imagen (Kernel image processing, 2017)

- **No linealidad (ReLU)**

ReLU (Rectified Linear Unit) es una operación de rectificación lineal que se aplica después del proceso de convolución. Se aplica pixel a pixel y reemplaza con un cero todos los valores negativos de en el mapa de características.

Su propósito es introducir no linealidad ya que casi toda la información del mundo real que se quiere que las CNN aprenda suele ser no lineal.

- **Sub-muestreo o Pooling**

Este proceso consiste en reducir la dimensionalidad de cada mapa de características, pero manteniendo la información más importante. Existen varios tipos como puede ser el máximo, la media o la suma. En el siguiente ejemplo se puede observar como a partir de un mapa de características de 4x4 aplicando un proceso de sub-muestreo de tipo máximo utilizando una ventana de 2x2 se obtiene la matriz resultante de 2x2:



Figura 11: Ejemplo de sub-muestreo (Karpthy, CS231n Convolutional Neural Networks for Visual Recognition, 2016)

- **Clasificación**

Los tres procesos anteriores pueden efectuarse varias veces obteniendo en cada iteración más mapas de características, pero cada vez más pequeños resultandos esto en la extracción de características útiles de las imágenes con la adición de no linealidad y reducción de dimensionalidad.

El paso final es utilizar una capa o capas completamente conectadas, esto significa que todas las neuronas de la capa anterior están conectadas a todas las neuronas de la capa siguiente. Como se ha explicado los procesos anteriores se han encargado de obtener las características principales de la imagen de entrada y el objetivo ahora es utilizar dichas características para correctamente clasificar la imagen entre los posibles valores de la capa de salida. (Karn, An Intuitive Explanation of Convolutional Neural Networks, 2016).

2.1.5 Técnicas de segmentación de las CNNs

Las redes neuronales convolucionales han ido mejorando año tras año especialmente desde 2012 cuando Alex Krizhevsky, Geoff Hinton e Ilya Sutskever ganaron ImageNet y desde entonces las CNNs han mejorado hasta el punto de superar a los humanos en dicho reto. (Alex Krizhevsky, 2012)

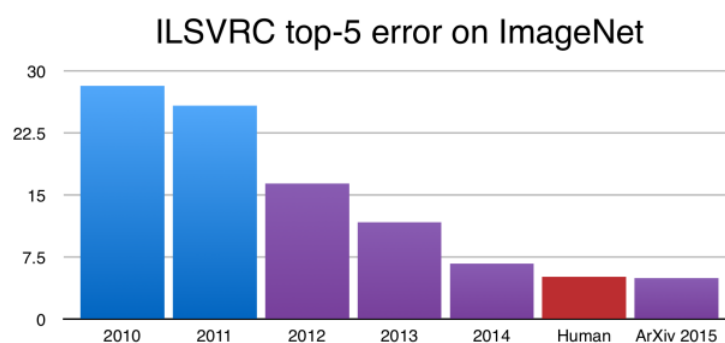


Figura 12: Evolución de la tasa de error ImageNet (Parthasarathy, 2017)

Pero en este reto ILSVRC (Imagenet Large Scale Visual Recognition Challenge) las imágenes utilizadas representan un único objeto el cual está enfocado y la tarea es clasificar dicho objeto. El problema es que en el mundo real las imágenes suelen representar múltiples objetos que se superponen entre sí con diferentes fondos por lo que la tarea de clasificar estas imágenes del mundo real y de ser capaces de separar donde empieza un objeto y empieza otro es mucho más compleja.

A continuación, se presentan una serie técnicas que se está utilizando para mejorar esta segmentación de los objetos de una imagen. Todas ellas parten de las premisas de una red neuronal convolucional.

- **R-CNN**

También llamadas redes neuronales convolucionales regionales. Su tarea consiste en dada una imagen de entrada identificar correctamente dónde se encuentran los objetos principales de ella. Se basa en utilizando un proceso de búsqueda selectiva crear una serie de rectángulos o posibles regiones que encuadren los principales objetos de la imagen, es decir, observa la imagen desde ventanas de diferentes tamaños y para cada tamaño intenta agrupar píxeles adyacentes por textura, color o intensidad para identificar objetos.

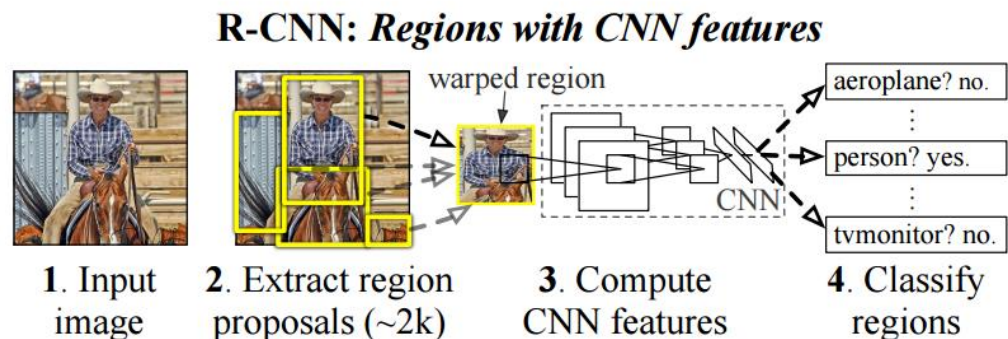


Figura 13: R-CNN

Una vez las regiones están creadas las convierte en rectángulos y las pasa a una versión modificada de AlexNet, la ganadora de ImageNet 2012.

En la capa final añade una capa de soporte vectorial (SVM: Support Vector Machine) que se encarga de clasificar si es un objeto y de serlo qué objeto es.

El último paso de la R-CNN es una vez identificado el objeto intentar ajustar el rectángulo que lo engloba a las dimensiones del propio objeto utilizando una regresión lineal para obtener el resultado final. (Malik, 2014).



Figura 14: Clasificación de objetos (Malik, 2014)

- **Fast R-CNN**

El problema de las R-CNNs es que es un proceso lento ya que requiere que cada propuesta de región de una imagen (unas 2000 normalmente) pase a través de la CNN.

Otro problema es que tiene que entrenar tres diferentes modelos: la CNN para generar las características de la imagen, el clasificador que predice la clase y la modelo de regresión para ajustar los rectángulos.

Rob Girshick que fue es también uno de los creadores de las R-CNN ideó en 2015 una forma de solventar estos problemas. Primeramente, se dio cuenta de muchas de las regiones propuestas se sobreponían haciendo que en el paso de la CNN se realizaran los mismos cálculos constantemente así que su propuesta fue realizar este paso una sola vez para la imagen y después compartir la información calculada para cada una de las propuestas.

Esta técnica se denomina RoIPool (Region of Interest Pooling) y reduce el número de pasos en la CNN de unos 2000 a 1.

La segunda mejora es juntar los tres modelos presentes en la R-CNN en uno solo reemplazando el clasificador SVM con una capa encima de la CNN para obtener una clasificación y a su vez ejecutar la regresión lineal de forma paralela. (Ross Girshick, 2015).

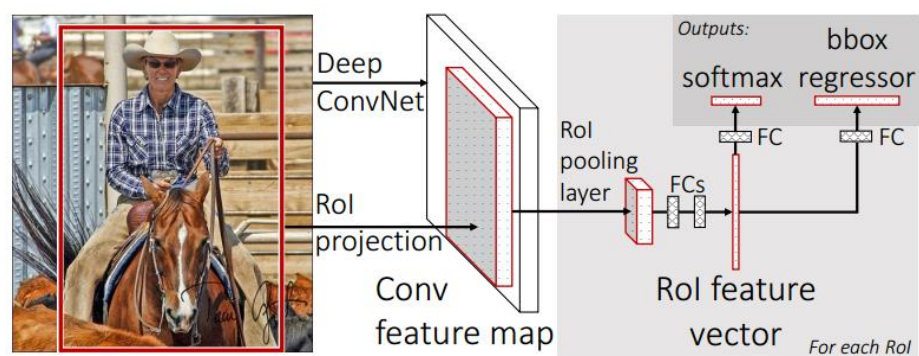


Figura 15: Fast R-CNN (Ross Girshick, 2015)

- **Faster R-CNN**

A pesar de los avances anteriores sigue habiendo un cuello de botella principal: las propuestas de regiones. La búsqueda selectiva es un proceso lento por lo que a mediados de 2015 un equipo de investigación de Microsoft encontró una manera de que este paso prácticamente se eliminara y lo llamaron Faster R-CNN. (Shaoqing Ren, 2016).

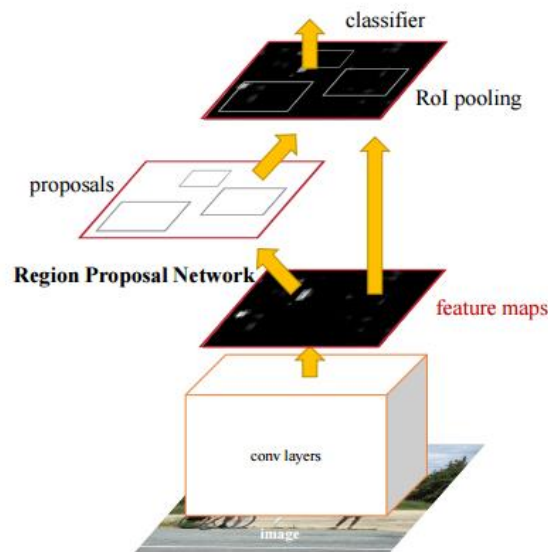


Figura 16: Faster R-CNN (Shaoqing Ren, 2016)

La idea es que las regiones propuestas dependen de características calculadas en el paso de la CNN así que se pueden reutilizar estos resultados para calcular las regiones.

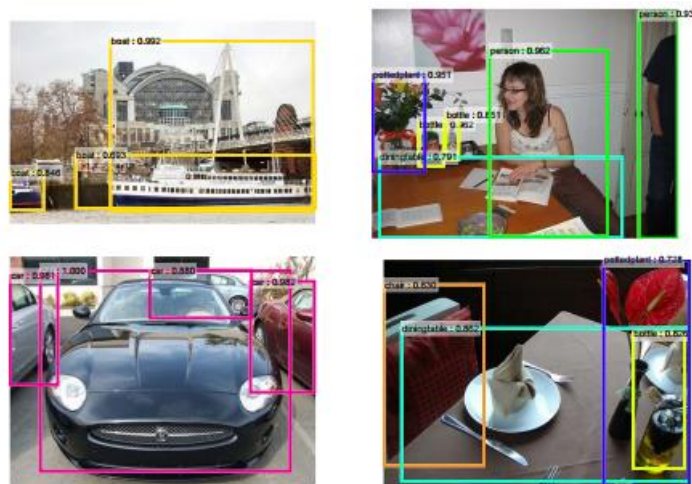


Figura 17: Clasificación de objetos con Faster R-CNN (Shaoqing Ren, 2016)

- **Mask R-CNN**

Hasta ahora hemos visto modelos que clasifican objetos de una imagen devolviendo una serie de rectángulos para cada objeto. Las Mask R-CNNs intentan mejorar este proceso localizando los píxeles exactos de una imagen que pertenecen a un objeto.

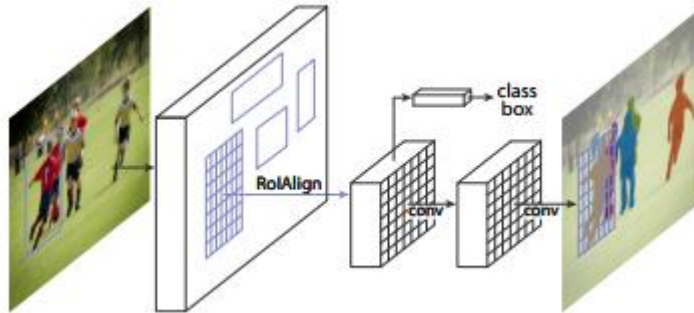


Figura 18: Mask R-CNN (Kaiming He, 2017)

La forma de realizar esto es, basándose en una Faster R-CNN calcular una máscara binaria en forma de matriz que diga si un pixel en concreto forma parte o no de un objeto.

Un problema que se encontraron fue que al ejecutar este proceso en el resultado final los objetos no estaban correctamente alineados del todo. Para solventarlo idearon un método llamado RoIAlign que consiste en sustituir el redondeo en la etapa RoIPool con una interpolación bilineal. (Kaiming He, 2017).

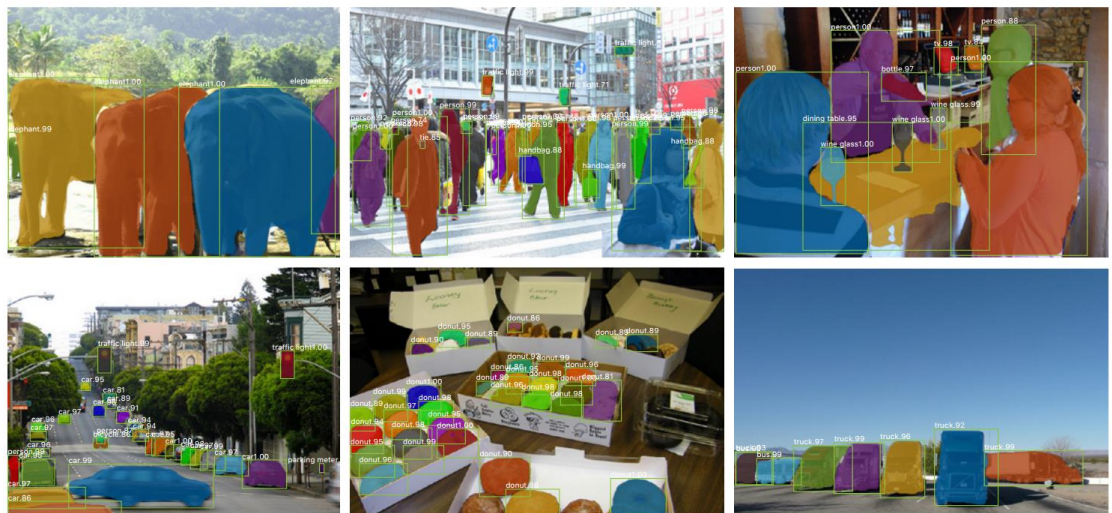


Figura 19: Segmentación por píxel (Kaiming He, 2017)

2.1.6 Herramientas comerciales para el reconocimiento facial

En los apartados anteriores se ha puesto en contexto la historia de la inteligencia artificial y se ha intentado explicar las distintas técnicas dentro de esta más eficaces actualmente en la

tarea de clasificación de objetos. En este último apartado se va a comparar las herramientas comerciales más populares que hacen uso de estas técnicas, pero centrándonos en los sistemas de reconocimientos faciales y de emociones.

En la siguiente tabla se muestra una comparativa de ellas mostrando de lo que son capaces y de lo que no:

	Detección facial	Reconocimiento facial (imagen)	Reconocimiento facial (video)	Clasificación de emociones	Detección de Edad y Género	Detección racial	Seguimiento multi-facial	SDK	API
Afectiva	✓	✗	✗	✓	✓	✗	✓	✓	✓
Amazon	✓	✓	✗	✓	✓	✗	✓	✗	✓
Google	✓	✗	✗	✓	✗	✗	✓	✗	✓
IBM	✓	✗	✗	✗	✓	✗	✓	✗	✓
Kairos	✓	✓	✓	✓	✓	✓	✓	✓	✓
Microsoft	✓	✓	✗	✓	✓	✗	✓	✗	✓
OpenCV	✓	✗	✓	✗	✗	✗	✓	✓	✗

Tabla 3: Comparativa de soluciones comerciales (Kairos, 2017)

2.2 Conclusiones

En este capítulo he intentado poner en contexto el estudio de la inteligencia artificial para después empezar a hablar de las redes neuronales y concretamente las redes neuronales convolucionales que se han demostrado en los últimos años ser muy eficaces a la hora de hacer clasificación de objetos en imágenes.

A su vez he comentado diferentes mejoras de este tipo de redes ideadas en los últimos años para finalizar haciendo una comparativa de las herramientas comerciales centradas en la clasificación facial.

Tras este análisis y teniendo en cuenta el problema que planteamos intentar resolver con este trabajo parece claro que la inteligencia artificial y concretamente las redes neuronales convolucionales son herramientas claves de las que se puede hacer uso para nuestro caso concreto.

En los capítulos siguientes paso a detallar mi aproximación a la resolución de este problema haciendo uso de las herramientas existentes.

CAPÍTULO 3

METODOLOGIA DE TRABAJO Y OBJETIVOS CONCRETOS

3.1 Objetivo general

Desarrollar una herramienta que facilite a las personas ciegas comprender información presentada en formato visual mediante el análisis y clasificación de las imágenes transformando los resultados de dicho análisis en formato sonoro permitiendo a su vez la interacción con el usuario a través de comandos vocales.

El tipo de información que se contempla analizar se centra en el reconocimiento facial de personas, así como de sus emociones, pero se pretende que la herramienta sea escalable a clasificar cualquier tipo de objeto.

3.2 Objetivos específicos

- Analizar una imagen de entrada de forma que permita:
 - Identificar la existencia de rostros.
 - Clasificar el sexo de los rostros encontrados.
 - Clasificar el estado emocional de los rostros encontrados.
 - Reconocer la identidad de los rostros cuando dicha identidad se encuentre almacenada en el sistema.
- A partir del análisis anterior visualizar en tiempo real los resultados de dicho análisis:
 - Señalizar visualmente los resultados del análisis sobre la imagen original.
 - Permitir obtener los resultados del análisis de forma auditiva en cualquier momento.
- Desarrollar un sistema que permita recibir comandos a través de la voz entendibles por la herramienta a través de los cuales se pueda:
 - Obtener información del análisis realizado sobre la imagen en cualquier momento.
 - Almacenar en el sistema la identidad de la persona reconocida en un momento dado para que el sistema sea capaz de reconocerla posteriormente.

3.3 Metodología de trabajo

La metodología que he considerado utilizar es de tipo incremental en la que en cada paso se va añadiendo funcionalidad a la herramienta hasta lograr todos los objetivos y obtener un producto mínimo funcional que cumpla todos ellos para después hacer una revisión iterativa del producto en búsqueda de mejoras en la eficiencia, portabilidad y posibles ampliaciones.

Para ello se han definido una serie de hitos que se muestran a continuación y que definen los pasos a seguir para el desarrollo del proyecto:

Hito	Descripción
H01	Análisis y definición de requisitos
H02	Investigación sobre el estado de arte y las herramientas y tecnologías de las que hacer uso para el desarrollo de la aplicación
H03	Seleccionar un entorno de desarrollo que se adecue a los requisitos
H04	Desarrollar un sistema que sea capaz de capturar imágenes para el posterior procesado y presentación de resultados
H05	Implantar un algoritmo de reconocimiento de caras en imágenes
H06	Implantar un algoritmo de clasificación de emociones
H07	Implantar un algoritmo de clasificación de género
H08	Integrar los algoritmos de reconocimiento y clasificación en el sistema y mostrar los resultados
H09	Desarrollar un sistema de entrada de comandos por teclado en el sistema
H10	Desarrollar un sistema para el soporte de múltiples idiomas para mostrar los resultados
H11	Implantar un sistema que permita transformar voz con soporte multilinguaje
H12	Implantar un sistema que permita transformar texto a voz
H13	Implementar un sistema que permita almacenar la identidad de una persona para ser recuperada posteriormente en la etapa de análisis de la imagen
H14	Desarrollar un sistema de entrada de comandos por voz en el sistema
H15	Revisión y evaluación
H16	Planteamiento de mejoras

Tabla 4: Hitos del proyecto

CAPÍTULO 4

DESARROLLO ESPECÍFICO DE LA CONTRIBUCIÓN

En este capítulo se detalla el desarrollo específico de la contribución comenzando con la identificación de requisitos del proyecto. En el siguiente apartado se pasa a describir exhaustivamente el proceso de desarrollo del producto y en el último apartado se describe el proceso de evaluación de la calidad.

4.1 Identificación de requisitos

En la fase de análisis del proyecto se capturan los requisitos del mismo. Éstos se dividen entre requisitos funcionales los cuales establecen el comportamiento del sistema y los no funcionales que se usan para juzgar la operación de un sistema en lugar de sus comportamientos específicos.

Cabe destacar que dada que el principal objetivo de este proyecto es servir de herramienta a las personas ciegas algunos requisitos que normalmente del ámbito de la accesibilidad que normalmente son clasificados como no funcionales son en este caso considerados funcionales y viceversa.

4.1.1 Requisitos funcionales

Requisito	Descripción
F01	El modelo de reconocimiento facial debe ser capaz de identificar caras de personas
F02	El modelo de clasificación de emociones debe ser capaz de identificar emociones de las personas identificadas
F03	El modelo de clasificación de género debe ser capaz de identificar el género de las personas identificadas
F04	La herramienta debe ser capaz de dada una imagen de entrada aplicar los diferentes modelos y obtener un resultado del análisis
F05	La herramienta debe ser capaz de informar de los resultados al usuario de forma auditiva
F06	La herramienta debe ser capaz de recibir instrucciones en forma de comandos de voz del usuario
F07	La herramienta debe permitir almacenar en el sistema la identidad de la persona identificada

F08	La herramienta debe ser capaz de reconocer si la identidad de la persona identificada pertenece a alguna de las almacenadas en el sistema y representar en la salida dicha identidad
-----	--

Tabla 5: Requisitos funcionales

4.1.2 Requisitos no funcionales

Requisito	Categoría	Descripción
NF01	Usabilidad	La herramienta debe ser capaz de recibir instrucciones a través del teclado
NF02	Usabilidad	La herramienta debe mostrar los resultados del análisis visualmente
NF03	Rendimiento	El tiempo de análisis de la imagen de entrada debe ser razonable
NF04	Accesibilidad	El sistema debe permitir la interacción en varios idiomas
NF05	Portabilidad	El sistema debe ser capaz de ejecutarse en varios entornos y aceptar imágenes de diferentes fuentes de entrada

Tabla 6: Requisitos no funcionales

4.2 Descripción detallada del producto resultante

4.2.1 Entorno de desarrollo

Se ha elegido Python (Python, 2017) como lenguaje de desarrollo ya que es un lenguaje muy utilizado en entornos de inteligencia artificial y existen paquetes todos los requisitos que este proyecto necesita.



Figura 20: Logo de Python (Python, 2017)

Concretamente se ha utilizado Python 3.5.2 y Anaconda 4.3 (Anaconda, 2017) que es una plataforma *open source* que permite procesamiento de grandes cantidades de información, análisis predictivo y que simplifica enormemente la gestión de los paquetes de Python.

Además, permite la creación y exportación de entornos concretos de desarrollo en Python lo que permite tener diferentes versiones de Python y/o de los paquetes instalados facilitando el desarrollo de diferentes aplicaciones evitando conflictos.

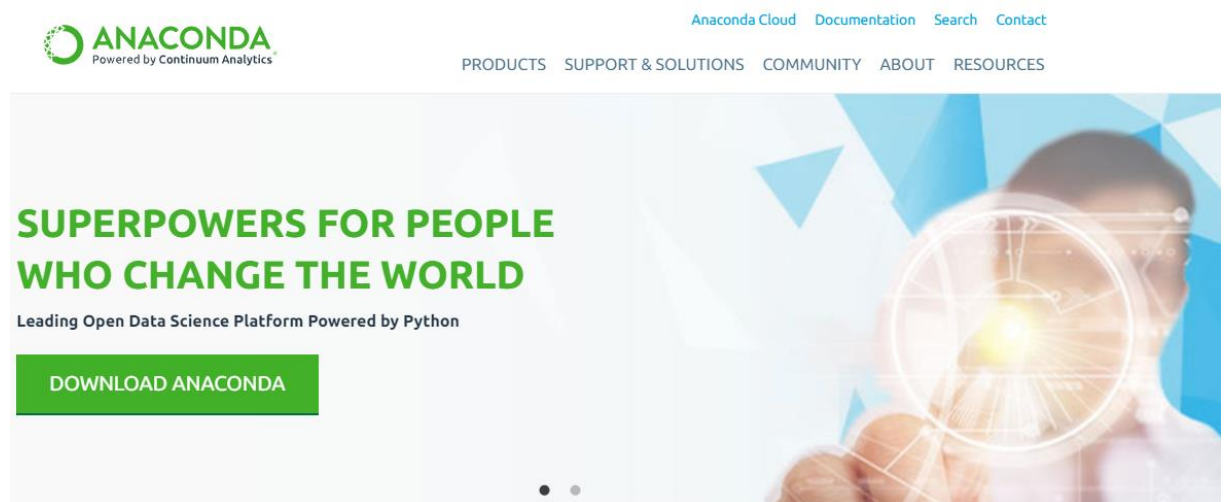


Figura 21: Anaconda (Anaconda, 2017)

Estos entornos que Anaconda proporciona pueden cargarse desde un archivo con extensión **yml** o pueden crearse a partir de estos archivos. Anaconda proporciona otras opciones de mantenimiento de entornos, pero sólo voy a nombrar estas dos.

A continuación, se muestra el contenido del archivo **environment.yml** que contiene información de los paquetes de Python necesarios para el funcionamiento del presente proyecto:

```
name: tfm
channels:
  - https://conda.anaconda.org/menpo
  - conda-forge
dependencies:
  - python==3.5.2
  - numpy
  - matplotlib
  - jupyter
  - opencv3
  - pillow
  - scikit-learn
  - scikit-image
  - scipy
```

```
- h5py
- eventlet
- flask-socketio
- seaborn
- pandas
- imageio=2.1.2
- pip:
  - moviepy
  - tensorflow==1.1.0
  - keras==2.0.3
  - statistics
  - face_recognition
  - face_recognition_models
  - dlib
  - pillow
  - pyaudio
  - google-api-python-client
  - google-cloud-speech==0.25.1
  - gTTS
  - pygame
  - SpeechRecognition
  - unidecode
```

En el Anexo 7.1 se muestra un archivo más detallado que incluye todas las dependencias que a su vez tienen los paquetes aquí mostrados, pero no es necesario utilizar este segundo archivo.

A partir del archivo ***environment.yml*** y con Anaconda instalado crear un entorno para que la herramienta desarrollada funcione correctamente es trivial. Se pueden encontrar instrucciones completas de manejo de entorno en Anaconda aquí: (Anaconda, 2017). A continuación, se muestran las instrucciones para crearlo en un entorno Windows:

Desde la línea de comandos se ejecuta:

```
conda env create -f environment.yml
```

Donde *environment* es la ruta al archivo ***environment.yml*** cuyo contenido se ha mostrado anteriormente. Después de ejecutarlo Anaconda comenzará a instalar la versión de Python y los paquetes especificados. En el archivo se configura que el entorno se llame ***tfm*** por lo que al concluir la instalación para activarlo se debe ejecutar:

```
activate tfm
```

Una vez con Python instalado, y el entorno configurado y activado el siguiente paso es elegir una IDE o herramienta para desarrollar el código de la aplicación. Si bien editores de textos como Sublime, Notepad++ o Visual Studio Code son perfectamente válidos en nuestro caso hemos utilizado Sypder y Jupyter Notebook los cuales vienen instalado con Anaconda. Para ejecutar Jupyter Notebook (Jupyter, 2017) desde la consola del sistema ejecutamos:

```
jupyter notebook
```

Lo cual nos abrirá en una ventana de nuestro navegador por defecto el editor, por defecto utilizando el puerto 8888 y en el directorio activo desde donde hemos lanzado Jupyter, mostrando nuestro navegador la siguiente dirección: <http://localhost:8888/tree> desde donde podemos acceder a las diferentes carpetas y archivos.

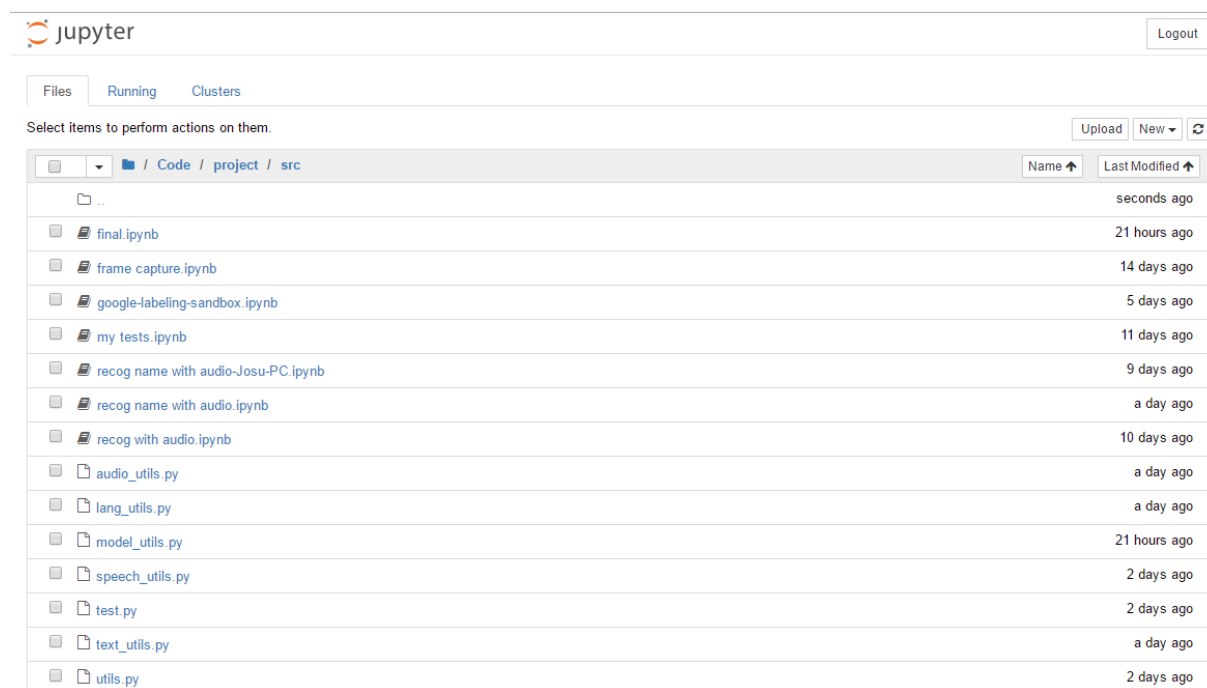
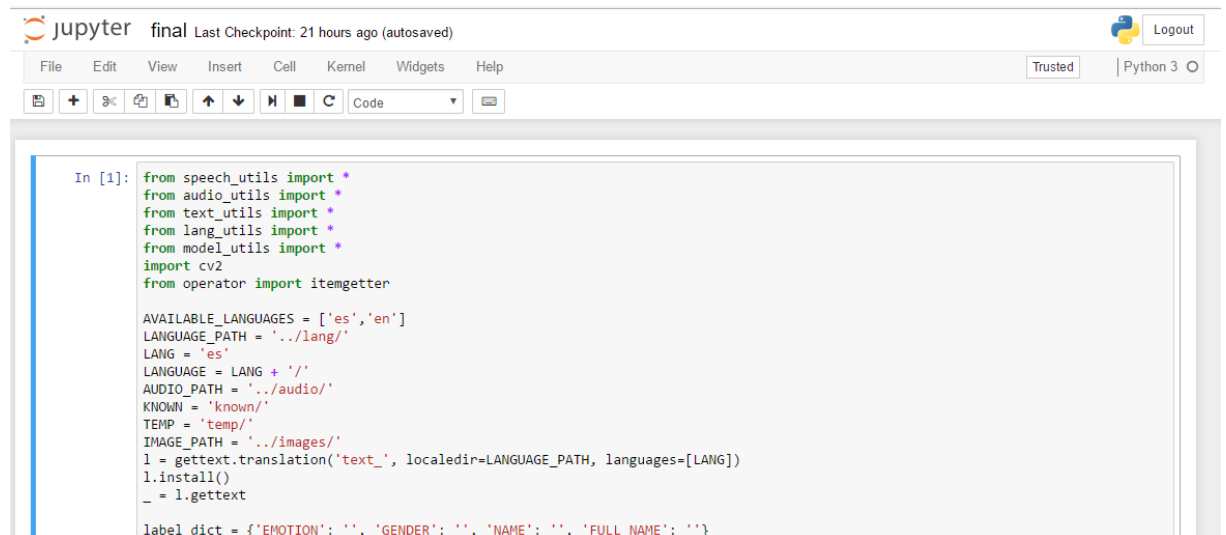


Figura 22: Jupyter Notebook en funcionamiento

Jupyter es capaz de abrir cualquier archivo de texto, pero para poder ejecutar código y que se muestre los resultados en sus celdas estos archivos deberán tener la extensión ***.ipynb***.

A continuación, se muestra un ejemplo del editor con un archivo de este tipo:



```
In [1]: from speech_utils import *
from audio_utils import *
from text_utils import *
from lang_utils import *
from model_utils import *
import cv2
from operator import itemgetter

AVAILABLE_LANGUAGES = ['es', 'en']
LANGUAGE_PATH = '../lang/'
LANG = 'es'
LANGUAGE = LANG + '/'
AUDIO_PATH = '../audio/'
KNOWN = 'known/'
TEMP = 'temp/'
IMAGE_PATH = '../images/'
l = gettext.translation('text_', localedir=LANGUAGE_PATH, languages=[LANG])
l.install()
_ = l.gettext

label_dict = {'EMOTION': '', 'GENDER': '', 'NAME': '', 'FULL_NAME': ''}
```

Figura 23: Ejemplo de archivo ipynb

Por último, sólo queda definir los dispositivos de entrada y salida del sistema.

- Dispositivos de entrada:
 - Imagen: en nuestro caso como dispositivo principal de captura de imágenes se va a utilizar una webcam que va a facilitar mucho el desarrollo, aunque se pretende que en el sistema final esto sea opcional y poder utilizar como fuente de entrada de imágenes capturas de pantalla.
 - Audio: como dispositivo de entrada de audio se va a utilizar el micrófono que viene integrado en la webcam. Esto se va a ut
 - Teclado: como método alternativo de entrada para los comandos principales se va a utilizar el teclado si bien es necesario recalcar que el método de activación de la entrada de comandos por audio es el teclado. Concretamente al pulsar la tecla espacio se activa la captura de comandos de audio siendo esta tecla la única imprescindible para que el sistema funcione correctamente.
 - Ratón: Se presupone que la herramienta puede hacer uso del ratón, aunque una vez en funcionamiento éste no es necesario.
- Dispositivos de salida:

- Pantalla: La herramienta muestra en pantalla la imagen de entrada capturada modificada para mostrar los resultados del análisis. Si bien mostrar los resultados visualmente no es necesario para el propósito del presente proyecto esto es muy útil a la hora de verificar el correcto funcionamiento del mismo.
- Audio: como dispositivo de salida de audio es necesario unos altavoces que permita al usuario obtener un *feedback* de los resultados del sistema, así como para proporcionar un método de comunicación con él.

Cabe destacar que los dispositivos propuestos son los necesarios para que la herramienta pueda ser utilizada en un ordenador que es donde se ha desarrollado y se han realizado las pruebas correspondientes pero la flexibilidad que ofrece Python hace que este sistema sea fácilmente a otros entornos como puede ser instalándose en una Raspberry Pi (Raspberry Pi, 2017) o similar que permita su portabilidad.

Por último, queda apuntar que para el correcto funcionamiento de la herramienta es necesario que esta tenga acceso a internet. Si bien ésta es capaz de funcionar sin una conexión su funcionalidad se ve seriamente limitada al hacerse uso de conexiones a API externas como por el cliente de conversación de Google (Google, 2017) y además se pretende en el futuro ampliar las funcionalidades de la herramienta por lo que no me cabe duda de que el acceso a internet es un requisito necesario.

Y hasta aquí la descripción del entorno de desarrollo del proyecto y de sus requerimientos. En los siguientes apartados se va a detallar las características de desarrollo de cada parte necesaria para el funcionamiento de la herramienta.

4.2.2 Modelos clasificatorios

En este apartado se va explicar los métodos utilizados para la que posiblemente sea la parte más importante de este proyecto que es el correcto análisis de la imagen de entrada para la identificación de caras y su clasificación emocional y de género. Como se puede desprender de lo ya comentado en el capítulo 2 los avances de los últimos años han hecho que esta tarea se simplifique enormemente existiendo multitud de alternativas si bien es un mundo en constante desarrollo y persistentemente aparecen nuevas ideas que mejoran en eficacia y eficiencia estas tareas.

Esto hace que sea difícil elegir una opción concreta por lo que se ha optado por opciones cuyo balance entre eficacia y rendimiento sea adecuado para el presente proyecto teniendo en

cuenta que nos interesa un análisis lo más rápido posible para aproximarlos al tiempo real considerando que nuestro principal método de entrada es una webcam por lo que queremos ser capaces de que cada imagen sea capturada, procesada y modificada sin sacrificar la fluidez de la salida.

En futuras ampliaciones del proyecto no se descartan cambiar los modelos utilizados para los diferentes apartados de clasificación para mejorar eficiencia o eficacia, pero por el momento se van a pasar a describir los sistemas actualmente utilizados.

4.2.2.1 Sistema de detección de caras

La detección de caras en una imagen es posiblemente la tarea más sencilla y rápida de todas las tareas que engloban el análisis de la imagen teniendo en cuenta que existen tecnologías con varios años ya por lo que su capacidad y fiabilidad está ampliamente constatada.

La tarea que queremos acometer es básicamente la siguiente: dada una imagen de entrada obtener una lista de las caras presentes en dicha imagen donde cada elemento de dicha lista nos proporcione la ubicación de la cara detectada de forma que con dicha información seamos capaces de enmarcar la cara o caras en la imagen original.

La forma en la que vamos a conseguir dicho objetivo es haciendo uso del sistema de clasificación de características basado en Haar para detección de objetos (Open CV, 2017).

Es un método propuesto por Paul Viola y Michael Jones en 2001 en su trabajo “Detección rápida de objetos usando cascadas impulsadas por características simples” (Paul Viola, 2001).

Se basa en un enfoque de aprendizaje de máquinas donde cada función cascada es entrenada con muchas imágenes positivas y negativas de lo que se pretende clasificar para después ser utilizada en otras imágenes. En nuestro caso concreto de detección de caras el algoritmo hace uso inicialmente de multitud de imágenes que contienen caras, así como de imágenes que no las contienen para entrenar al clasificador. Cada característica es un único valor obtenido de restar la suma de los píxeles debajo del rectángulo blanco de la suma de píxeles debajo del rectángulo negro:

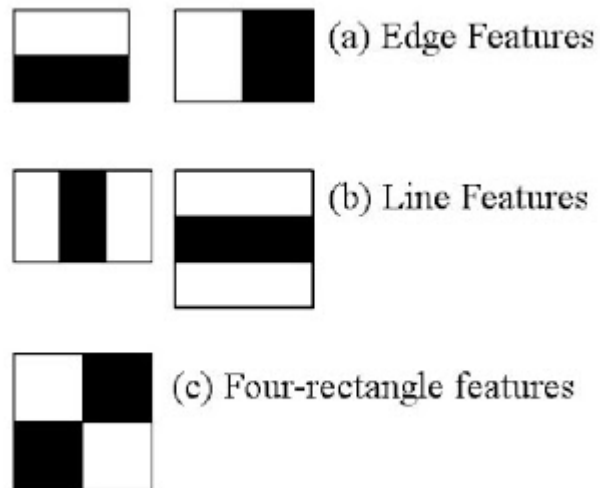


Figura 24: Características Haar (Open CV, 2017)

Posteriormente todos los posibles tamaños y localizaciones de cada *kernel* son usados para calcular multitud de características haciendo que la capacidad computacional sea bastante grande en función del tamaño de la imagen por lo que para solventar este problema se simplifica el cálculo de la suma de píxeles y a como de grande puede ser el número de píxeles a una operación que solo implica a cuatro píxeles. Además, se utiliza Adaboost (Schapire, 1997) que implica que no se le dé la misma importancia a aquellas características calculadas que sean irrelevantes siendo esta la mayoría.

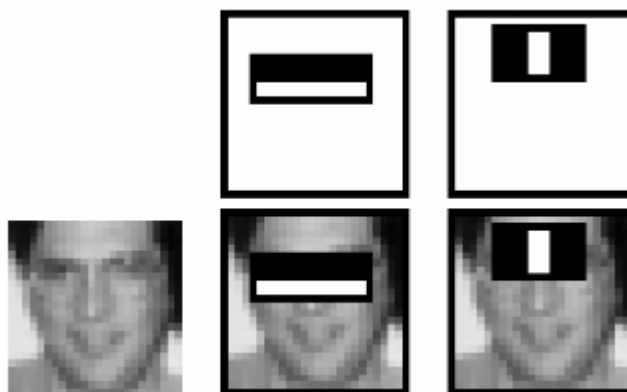


Figura 25: Ejemplo Haar (Open CV, 2017)

En otras palabras, el clasificador final se va a componer de la media de la suma de una serie de clasificadores débiles o *weak* que básicamente son clasificadores que por si solos no son capaces de clasificar correctamente, pero juntos crean un algoritmo de clasificación robusto.

Por último, se introduce el concepto de clasificadores en cascada que lo que hace es que en vez de aplicar todas las características detectadas en una imagen las agrupe en diferentes pasos de clasificación y los aplique uno a uno. Y esto es básicamente la explicación del funcionamiento del algoritmo que vamos a utilizar para la detección de caras, a continuación, vamos a explicar cómo hacer uso de éste para nuestra tarea.

OpenCV (Open CV, 2017) es una librería *open source* que se puede instalar como paquete en Python y que está enfocado al procesamiento de imágenes por ordenador en tiempo real. OpenCV permite el uso de *Haar Cascades* tanto para entrenamiento como para clasificación. En el caso de entrenamiento se genera un archivo *.xml* que puede ser utilizado posteriormente para la clasificación. En nuestro proyecto hacemos uso de uno de estos archivos, concretamente uno entrenando para detectar caras frontales. La ubicación de este archivo en el proyecto es

```
./models/haarcascade_frontalface_default.xml
```

A continuación, se muestra una parte del mismo:

```

45 <opencv_storage>
46 <cascade type_id="opencv-cascade-classifier"><stageType>BOOST</stageType>
47 <featureType>HAAR</featureType>
48 <height>24</height>
49 <width>24</width>
50 <stageParams>
51 <maxWeakCount>211</maxWeakCount></stageParams>
52 <featureParams>
53 <maxCatCount>0</maxCatCount></featureParams>
54 <stageNum>25</stageNum>
55 <stages>
56 <_>
57 <maxWeakCount>9</maxWeakCount>
58 <stageThreshold>-5.0425500869750977e+00</stageThreshold>
59 <weakClassifiers>
60 <_>
61 <internalNodes>
62 0 -1 0 -3.1511999666690826e-02</internalNodes>
63 <leafValues>
64 2.0875380039215088e+00 -2.2172100543975830e+00</leafValues></_>
65 <_>
66 <internalNodes>
67 0 -1 1 1.2396000325679779e-02</internalNodes>
68 <leafValues>
69 -1.8633940219879150e+00 1.3272049427032471e+00</leafValues></_>
70 <_>
71 <internalNodes>
72 0 -1 2 2.1927999332547188e-02</internalNodes>
73 <leafValues>
74 -1.5105249881744385e+00 1.0625729560852051e+00</leafValues></_>

```

Figura 26: Ejemplo archivo xml generado

En el repositorio GitHub de OpenCV se muestra un ejemplo de cómo se utilizaría un archivo de este tipo para la detección de caras (Open CV, 2016):

Y a continuación muestro un ejemplo del concreto obtenido de la página de documentación de OpenCV en el que se hace uso de un algoritmo de detección de caras y otro de detección de ojos para después mostrar los resultados en la imagen original.

```
import numpy as np
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
img = cv2.imread('sachin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)

for (x,y,w,h) in faces:
    cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (0,255,0), 2)
cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

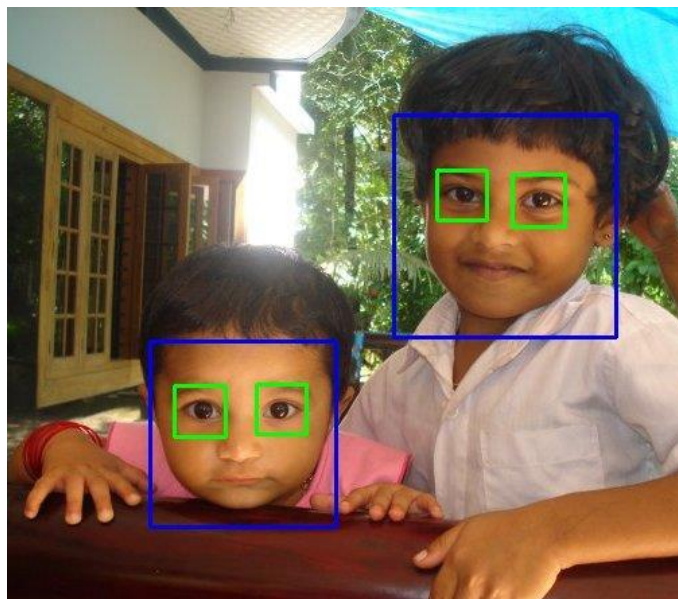


Figura 27: Resultado de la detección (Open CV, 2017)

4.2.2.2 Sistema de reconocimiento de caras

El sistema de reconocimiento de caras se va a utilizar para permitir al usuario dada una cara previamente detectada almacenar la identidad de la persona para que posteriormente dicha identidad sea mostrada en la salida de la herramienta.

Para ello se va a hacer uso del paquete “Face Recognition” disponible para instalar en Python (face_recognition 0.1.6, 2017).

Este paquete permite reconocer y manipular imágenes usando la librería “dlib” (Dlib C++ Library, 2017) la cual proporciona una serie de herramientas desarrolladas en C++ principalmente algoritmos de aprendizaje de máquinas y que tiene una precisión de 99,38% a la hora de etiquetar caras con la base de datos “Labeled Faces in the Wild” (University of Massachusetts, 2017). También proporciona detección de caras, pero en nuestro caso sólo vamos a utilizar la parte de reconocimiento ya que a pesar de tener una precisión mayor también es bastante más lento que nuestro método utilizando “Haar Cascades” de OpenCV. Sobre este tema se hablará más adelante en el apartado de integración.

Así que básicamente el funcionamiento en nuestro caso concreto es el siguiente: Dada una parte de una imagen original la cual representa una cara detectada (realmente se puede cargar la imagen completa y dejar que el algoritmo detecte las caras pero haciéndolo así hacemos que el sistema sea más rápido) dicha parte de la imagen se carga utilizando la librería “Face Recognition” devolviendo una serie de puntos de referencia de la cara en forma de posiciones que vienen a ser el equivalente a una huella dactilar que nos van a servir para posteriormente identificar a una persona concreta.

A continuación, se muestran unos ejemplos proporcionados en la página de documentación de la librería para facilitar el entendimiento, pero en el apartado de integración explicaré más en detalle cómo se ha utilizado en este proyecto.

Dada una imagen de entrada se obtiene una identificación de los rasgos fáciles de la cara como puede ser los ojos, la nariz o la boca:

```
import face_recognition
image = face_recognition.load_image_file("your_file.jpg")
face_landmarks_list = face_recognition.face_landmarks(image)
```

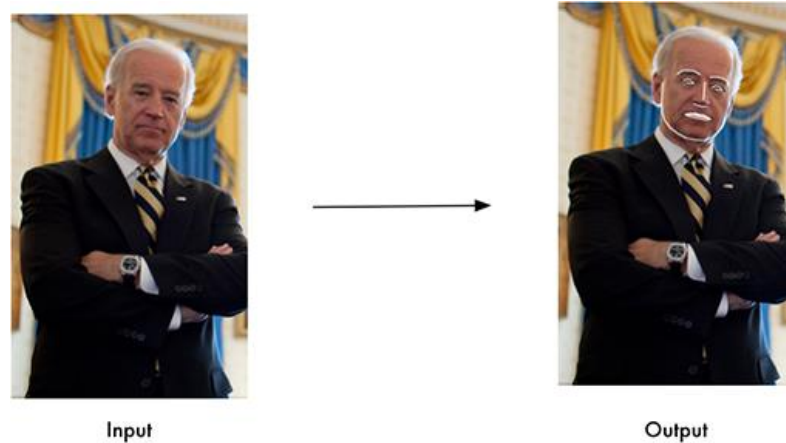


Figura 28: Ejemplo de reconocimiento de rasgos faciales

Y en este otro ejemplo se ve como primeramente se carga una imagen perteneciente al político estadounidense Joe Biden. A partir de esa imagen se obtienen las marcas características de su cara y posteriormente se utiliza ese resultado para comparar con una imagen desconocida. El algoritmo devolverá si la imagen desconocida contiene la cara de Joe Biden o no:

```
import face_recognition
known_image = face_recognition.load_image_file("biden.jpg")
unknown_image = face_recognition.load_image_file("unknown.jpg")

biden_encoding = face_recognition.face_encodings(known_image)[0]
unknown_encoding = face_recognition.face_encodings(unknown_image)[0]

results = face_recognition.compare_faces([biden_encoding], unknown_encoding)
```

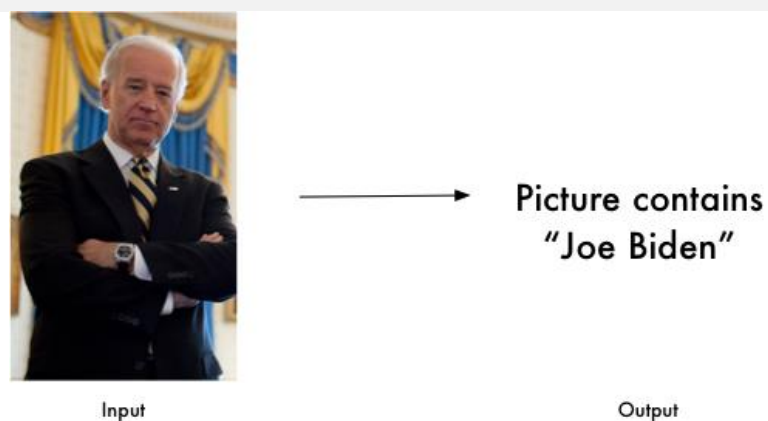


Figura 29: Ejemplo de reconocimiento de identidades

4.2.2.3 Sistema de clasificación de emociones

Tanto para el sistema de clasificación de emociones como de género se va a utilizar la librería “face_classification” disponible en (Arriaga, face_classification, 2017) y cuyo entrenamiento se pasa a describir a continuación.

Para obtener el modelo de clasificación de emociones la librería escogida utiliza el dataset fer2013 proporcionado en el año 2013 en la plataforma Kaggle para concursos de análisis y modelo predictivo en el que empresas e investigadores publican su información para que después gente de todo el mundo compita para obtener los mejores modelos. El dataset se puede obtener aquí: (Kaggle, 2013)

Básicamente el dataset consiste en un archivo csv de 28709 ejemplos de entrenamiento y 3589 de prueba divididos en tres columnas en las que la primera columna representa la emoción anotada, la segunda imágenes de 48x48 píxeles en blanco y negro y la tercera especifica si corresponde a imagen de entrenamiento o de prueba. Los posibles valores de cada imagen, es decir, los valores representados en la primera columna son los siguientes:

0. Enfado (Angry)
1. Asco (Disgust)
2. Miedo (Fear)
3. Alegría (Happy)
4. Tristeza (Sad)
5. Sorpresa (Surprise)
6. Neutral (Neutral)

Para la fase de entrenamiento la librería utilizada opta por utilizar Keras (Keras, 2017) que es una librería de redes neuronales *open source* escrita en Python que se ejecuta encima de otras librerías facilitando el desarrollo. En el caso que nos ocupa ésta se ejecuta encima de TensorFlow (Google, 2017) que fue desarrollada por Google para cubrir sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones análogos al aprendizaje y razonamiento que los seres humanos utilizamos.

A continuación, se muestra el código de la función `simple_CNN` donde se configura una red neuronal convolucional para el entrenamiento de algoritmos de clasificación de emociones. (Arriaga, models.py, 2017)

```
from keras.layers import Activation, Convolution2D, Dropout
from keras.layers import AveragePooling2D, BatchNormalization
from keras.layers import GlobalAveragePooling2D
from keras.models import Sequential

def simple_CNN(input_shape, num_classes):

    model = Sequential()
    model.add(Convolution2D(filters=16, kernel_size=(7, 7), padding='same',
                            name='image_array', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=16, kernel_size=(7, 7), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=128, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
```

```

model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
model.add(Dropout(.5))

model.add(Convolution2D(filters=256, kernel_size=(3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Convolution2D(filters=num_classes, kernel_size=(3, 3),
padding='same'))
model.add(GlobalAveragePooling2D())
model.add(Activation('softmax',name='predictions'))
return model

```

La configuración completa de la red se puede observar a continuación en el que se detalla cada capa y su tipo, la forma de salida de la capa y el número de parámetros.

Por ejemplo, inicialmente cada imagen tiene unas dimensiones de 48x48x1, es decir, 48 píxeles de alto y 48 de ancho y solo una dimensión ya que la imagen está en blanco y negro. La primera convolución transforma esta imagen de 48x48x1 a 48x48x16 al utilizarse 16 filtros.

Posteriormente se ejecuta otra convolución con 32 filtros lo que hace que las dimensiones obtenidas pasen a ser 24x24x32. En la siguiente pasa a ser de 12x12x64, luego 6x6x128, 3x3x256... Es decir, en cada convolución se reduce el ancho y alto de la entrada a la mitad, pero se duplica la profundidad. Entre convolución y convolución se realizan otras acciones como la normalización, *pooling* y *dropout* que se utiliza para prevenir el sobreajuste de la red. (Nitish Srivastava, 2014).

En la última capa se conecta la salida de la red a los posibles valores de salida obteniendo el modelo de clasificación de emociones.

Layer (type)	Output Shape	Param #
image_array (Conv2D)	(None, 48, 48, 16)	800
batch_normalization_1 (Batch Normalization)	(None, 48, 48, 16)	64
conv2d_1 (Conv2D)	(None, 48, 48, 16)	12560
batch_normalization_2 (Batch Normalization)	(None, 48, 48, 16)	64
activation_1 (Activation)	(None, 48, 48, 16)	0

average_pooling2d_1	(Average (None, 24, 24, 16)	0
dropout_1	(Dropout) (None, 24, 24, 16)	0
conv2d_2	(Conv2D) (None, 24, 24, 32)	12832
batch_normalization_3	(Batch (None, 24, 24, 32)	128
conv2d_3	(Conv2D) (None, 24, 24, 32)	25632
batch_normalization_4	(Batch (None, 24, 24, 32)	128
activation_2	(Activation) (None, 24, 24, 32)	0
average_pooling2d_2	(Average (None, 12, 12, 32)	0
dropout_2	(Dropout) (None, 12, 12, 32)	0
conv2d_4	(Conv2D) (None, 12, 12, 64)	18496
batch_normalization_5	(Batch (None, 12, 12, 64)	256
conv2d_5	(Conv2D) (None, 12, 12, 64)	36928
batch_normalization_6	(Batch (None, 12, 12, 64)	256
activation_3	(Activation) (None, 12, 12, 64)	0
average_pooling2d_3	(Average (None, 6, 6, 64)	0
dropout_3	(Dropout) (None, 6, 6, 64)	0
conv2d_6	(Conv2D) (None, 6, 6, 128)	73856
batch_normalization_7	(Batch (None, 6, 6, 128)	512
conv2d_7	(Conv2D) (None, 6, 6, 128)	147584
batch_normalization_8	(Batch (None, 6, 6, 128)	512
activation_4	(Activation) (None, 6, 6, 128)	0
average_pooling2d_4	(Average (None, 3, 3, 128)	0

dropout_4 (Dropout)	(None, 3, 3, 128)	0
<hr/>		
conv2d_8 (Conv2D)	(None, 3, 3, 256)	295168
<hr/>		
batch_normalization_9 (Batch Normalization)	(None, 3, 3, 256)	1024
<hr/>		
conv2d_9 (Conv2D)	(None, 3, 3, 7)	16135
<hr/>		
global_average_pooling2d_1 (Global Average Pooling)	(None, 7)	0
<hr/>		
predictions (Activation)	(None, 7)	0
<hr/>		
=====		
Total params: 642,935		
Trainable params: 641,463		
Non-trainable params: 1,472		
<hr/>		

El algoritmo se entrena durante 1000 iteraciones con un tamaño de bloque de 128 y un optimizador *Adam* (Diederik P. Kingma, 2017) que calcula tasas de aprendizaje adaptativas para cada parámetro llegando a una precisión del 66% en la clasificación de emociones. El resultado no es impresionante pero después de efectuar pruebas con el modelo resultante la satisfacción con él es correcta, pero queda pendiente mejorarlo en posibles ampliaciones.

A continuación, se muestra los resultados de las cinco primeras iteraciones del algoritmo de entrenamiento:

```
Epoch 1/1000
28672/28709 [=====>.] - ETA: 0s - loss: 1.7308 - acc:
0.3095
Epoch 00000: val_acc improved from -inf to 0.27097, saving model to
../trained_models/emotion_models/simple_CNN.00-0.27.hdf5
28709/28709 [=====] - 516s - loss: 1.7308 - acc: 0.3096
- val_loss: 1.7873 - val_acc: 0.2710

Epoch 2/1000
28672/28709 [=====>.] - ETA: 0s - loss: 1.5813 - acc:
0.3797
Epoch 00001: val_acc improved from 0.27097 to 0.38451, saving model to
../trained_models/emotion_models/simple_CNN.01-0.38.hdf5
```

```

28709/28709 [=====] - 522s - loss: 1.5813 - acc: 0.3798
- val_loss: 1.6210 - val_acc: 0.3845

Epoch 3/1000
28672/28709 [=====>.] - ETA: 0s - loss: 1.5108 - acc:
0.4098
Epoch 00002: val_acc improved from 0.38451 to 0.42770, saving model to
../trained_models/emotion_models/simple_CNN.02-0.43.hdf5
28709/28709 [=====] - 498s - loss: 1.5108 - acc: 0.4098
- val_loss: 1.5411 - val_acc: 0.4277

Epoch 4/1000
28672/28709 [=====>.] - ETA: 0s - loss: 1.4556 - acc:
0.4336
Epoch 00003: val_acc improved from 0.42770 to 0.43703, saving model to
../trained_models/emotion_models/simple_CNN.03-0.44.hdf5
28709/28709 [=====] - 496s - loss: 1.4558 - acc: 0.4335
- val_loss: 1.4892 - val_acc: 0.4370

Epoch 5/1000
28672/28709 [=====>.] - ETA: 0s - loss: 1.4136 - acc:
0.4524
Epoch 00004: val_acc improved from 0.43703 to 0.49429, saving model to
../trained_models/emotion_models/simple_CNN.04-0.49.hdf5
28709/28709 [=====] - 515s - loss: 1.4134 - acc: 0.4525
- val_loss: 1.3272 - val_acc: 0.4943

```

Se puede observar que en la primera iteración el modelo es capaz de clasificar correctamente solo el 27% de las imágenes, pero en la quinta iteración la presión aumenta al 43%. En nuestra herramienta vamos a utilizar el modelo obtenido en la iteración 985 y que cuenta con una presión del 66%. Este modelo se encuentra en

```
../models/emotion/simple_CNN.985-0.66.hdf5
```

4.2.2.4 Sistema de clasificación de género

Como en el sistema anterior se hace uso de la librería “face_classification” pero en este caso para la obtención de un modelo clasificatorio del género.

En este caso el dataset utilizado procede de imágenes catalogadas con género y edad de IMDB y Wikipedia. Concretamente se utiliza la versión que tiene las caras recortadas para

facilitar el entrenamiento. En nuestro caso vamos a hacer uso de las etiquetas de género únicamente siendo los valores los siguientes:

0. Hombre (Male)

1. Mujer (Female)

El dataset está disponible aquí: (Rasmus Rothe, 2015)

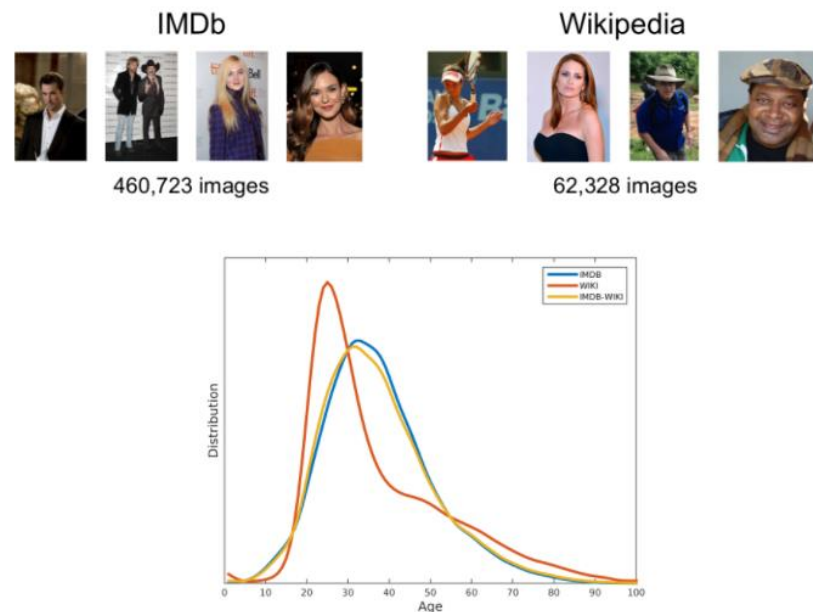


Figura 30: Origen de los datos utilizados (Rasmus Rothe, 2015)

Una vez más hacemos uso del método Simple_CNN detallado anteriormente entrenando durante 1000 iteraciones con un tamaño de bloque de 32 y un optimizador *Adam* obteniendo una precisión del 96% en la iteración 81 y es este modelo el que vamos a utilizar el cual se encuentra en en

```
./models/gender/simple_CNN.81-0.96.hdf5
```

4.2.3 Sistema de conversión voz a texto

Dada la naturaleza del proyecto el cual pretende facilitar la interacción de las personas ciegas está claro que es necesario algún sistema de comunicación con el usuario que no requiera hacer uso de la vista. En este caso se ha optado por utilizar un sistema que a través de un micrófono capture las instrucciones que el usuario quiere efectuar en el sistema. El detalle completo de los comandos disponibles se detallará más adelante en el apartado opciones del

sistema. En este apartado voy a comentar el proceso utilizado para satisfacer este requerimiento.

La herramienta elegida inicialmente para implementar esta funcionalidad fue la librería SpeechRecognition (Anthony Zhang, 2014) la cual proporciona un interfaz común de acceso a diferentes librerías de reconocimiento de voz como CMU Sphinx, Google Speech Recognition, Google Cloud Speech API, etc. Si bien esta librería es capaz de funcionar offline para alguna de sus opciones su principal carencia desde mi punto de vista es que al unificar de alguna manera el acceso a las diferentes librerías disponibles ha tenido que eliminar acceso a algunas funcionalidades que para este proyecto considero importantes como son la posibilidad de especificar el idioma utilizado lo que hace que la calidad de la transcripción no sea la deseable para los idiomas que no sean inglés.

Por esta razón al final he decidido utilizar el Google Cloud Speech API (Google, 2017) al que la única pega que le puedo poner es que es necesario una conexión a internet pero que sus ventajas y calidad justifican plenamente este ligero problema.

A pesar de todos decidí mantener las dos integraciones siendo la principal la de Google y en caso de fallo de esta o incapacidad para obtener una transcripción se utiliza la librería SpeechRecognition. De este modo se solventa el problema de la necesidad de una conexión a internet, aunque la calidad de las transcripciones empeora.

A continuación, se muestra el código de la función `transcript_audio` que recibe como parámetros de entrada la ubicación del archivo de audio a transcribir, el lenguaje utilizado (“es” para español, “en” para inglés) y por último un parámetro que determina si se usa la librería de Google Cloud o no.

```
def transcript_audio(filepath, language, use_cloud):
    transcript = '##NONE##'
    # The name of the audio file to transcribe
    file_name = os.path.join(os.path.dirname(''), filepath)

    if use_cloud:
        try:
            # Instantiates a client
            speech_client =
speech.Client.from_service_account_json(GOOGLE_CLOUD_SPEECH_CREDENTIALS_PATH)

            # Loads the audio into memory
```

```

        with io.open(file_name, 'rb') as audio_file:
            content = audio_file.read()
            sample = speech_client.sample(
                content,
                source_uri=None,
                encoding='LINEAR16',
                sample_rate_hertz=16000)

            # Detects speech in the audio file
            alternatives = sample.recognize(language)

            if (len(alternatives)>0):
                transcript = alternatives[0].transcript
        except Exception as e:
            print(e)

    if (transcript == '##NONE##'):
        try:
            r = sr.Recognizer()
            with sr.AudioFile(file_name) as source:
                audio = r.record(source)

            # for testing purposes, we're just using the default API key
            # to use another API key, use `r.recognize_google(audio,
key="GOOGLE_SPEECH_RECOGNITION_API_KEY", show_all=True)`
            # instead of `r.recognize_google(audio, show_all=True)`
            alternatives = r.recognize_google(audio, show_all=False)
            if (len(alternatives)>0):
                transcript = alternatives
        except sr.UnknownValueError:
            print("Google Speech Recognition could not understand audio")
        except sr.RequestError as e:
            print("Could not request results from Google Speech Recognition
service; {0}".format(e))

    return transcript

```

4.2.4 Sistema de conversión texto a voz

Del mismo modo que en el apartado anterior indicábamos la necesidad de tener un sistema que hiciera una conversión de voz a texto para ser capaces de analizar comandos de voz

también necesitamos una herramienta que nos permita transformar texto a audio para que el sistema sea capaz de comunicarse de manera dinámica con el usuario sin necesidad de tener grabado con antelación todas las posibles opciones.

Esto es de especial relevancia en nuestro proyecto cuando el usuario desea almacenar en el sistema la identidad de una persona conocida y que se explicará en el siguiente apartado con detalle.

La librería elegida ha sido gTTS (Durette, 2016) que es una interfaz Python para el Text To Speech API de Google que además soporta diferentes idiomas por lo que como en el caso anterior nos es muy útil. El problema que tiene es que el archivo de audio generado es en formato mp3 y para entender por qué es un problema hay que saber que en el desarrollo del proyecto hasta este momento los archivos de audio con los que se estaba trabajando eran archivos *wav* que funcionan muy bien y se integran perfectamente con los sistemas de voz a texto a utilizados y existen múltiples opciones de reproducción de archivos de este tipo en Python. Para reproducir archivos mp3 en cambio no existen tantas opciones de librerías y normalmente requieren instalar a su vez otras librerías que soporten compresión de audio como *ffmpeg* y que a menudo no son fácilmente instalables por problemas de conflictos.

Después de varios intentos con diversas opciones para reproducir los archivos mp3 al final opté por utilizar la librería *Pygame* (Pete Shinnars, 2017) que proporciona un envoltorio para librerías SDL multimedia (Simple DirectMedia Layer, 2017) para reproducción de audio, video y captura de teclado, ratón entre otros.

Primeramente, se va a mostrarla función `play_audio` que reproduce un archivo mp3 utilizando la librería *Pygame*:

```
import pyaudio
import wave
import time
import sys
import pygame as pg

def play_audio(audio_file, volume=0.8):
    '''
    stream music with mixer.music module in a blocking manner
    this will stream the sound from disk while playing
    '''
    # set up the mixer
```

```

freq = 44100      # audio CD quality
bitsize = -16    # unsigned 16 bit
channels = 2      # 1 is mono, 2 is stereo
buffer = 2048    # number of samples (experiment to get best sound)
pg.mixer.init()
# volume value 0.0 to 1.0
pg.mixer.music.set_volume(volume)
clock = pg.time.Clock()
try:
    pg.mixer.music.load(audio_file)
    print("Audio file {} loaded!".format(audio_file))
except pg.error:
    print("File {} not found! ({}).format(audio_file, pg.get_error())
    return
pg.mixer.music.play()
while pg.mixer.music.get_busy():
    # check if playback has finished
    clock.tick(30)

```

Y a continuación se muestra el código para grabar un archivo de audio, primero en español y después en inglés:

```

tts = gTTS(text='Lo siento, no te he entendido', lang='es', slow=False)
tts.save("../project/lang/es/speech/sorry_understand.mp3")

tts = gTTS(text='Sorry, I didn't get that', lang='en', slow=False)
tts.save("../project/lang/en/speech/sorry_understand.mp3")

```

Estos archivos pueden estar pregrabados o crearse al vuelo, aunque obviamente esto lleva más tiempo por lo que por eficiencia se han pregrabado todas las opciones en la medida de lo posible.

4.2.5 Integración de los modelos

Una vez disponemos de los cuatro modelos de detección y clasificación es el momento de integrarlos en nuestro sistema. El proceso general se describe en el siguiente diagrama de flujo donde primeramente se captura una imagen para después hacer uso del modelo de detección de caras. En caso de detectarse alguna cara cada una de ellas se procesa y se modifica la imagen original para mostrar los resultados del análisis. Posteriormente se

comprueba se hay algún tipo de interacción por parte del usuario y de haberla se procesa. Para finalizar se muestra la imagen modificada y se inicia el proceso otra vez.

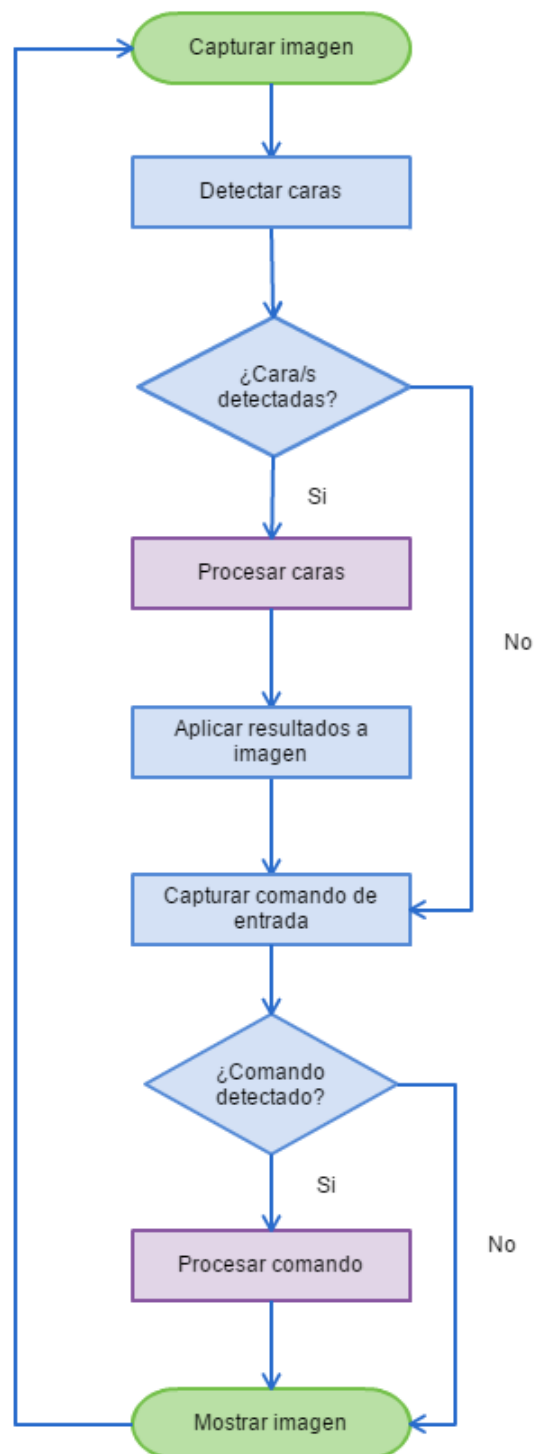


Figura 31: Diagrama de flujo de la herramienta

4.2.5.1 Inicialización

Antes de empezar a procesar imágenes se debe inicializar el sistema que consiste en configurar los parámetros iniciales, la localización e idioma que se utilizará por defecto en la aplicación además de cargarse los modelos de detección y clasificatorios a utilizar para ser utilizados posteriormente en la etapa de procesamiento. El flujo es el siguiente:



Figura 32: Diagrama de flujo de la inicialización

Se muestra a continuación el código correspondiente a este paso donde se puede apreciar entre otras cosas que se configuran dos idiomas (español e inglés) y se cargan los archivos procedentes del entrenamiento de los modelos de clasificación.

```
from speech_utils import *
from audio_utils import *
from text_utils import *
from lang_utils import *
from model_utils import *
```

```
import cv2
from operator import itemgetter

# Variables
ENCODING_FREQ = 10
encoding_count = 0
last_faces_count = 0
face_encodings = []
predictions = []
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.namedWindow('main')
emotion_label_window = []
gender_label_window = []
last_faces = []
label_dict = {'EMOTION': '', 'GENDER': '', 'NAME': '', 'FULL_NAME': ''}

# Language and localization
AVAILABLE_LANGUAGES = ['es', 'en']
LANGUAGE = 'es'
LANGUAGE_PATH = '../lang/'
AUDIO_PATH = '../audio/'
IMAGE_PATH = '../images/'

lang_helper = Lang_Helper(AVAILABLE_LANGUAGES, LANGUAGE_PATH, AUDIO_PATH,
IMAGE_PATH, LANGUAGE)

# Models
model_helper = Model_Helper('../models/face/haarcascade_frontalface_default.xml',
                             '../models/emotion/simple_CNN.530-0.65.hdf5',
                             '../models/gender/simple_CNN.81-0.96.hdf5',
                             AUDIO_PATH, IMAGE_PATH)

# Input image
video_capture = cv2.VideoCapture(0)
```

4.2.5.2 Detección de caras

El flujo de este proceso es el siguiente:



Figura 33: Diagrama de flujo de la detección de caras

En cada iteración se captura una imagen de entrada y se inicializan una serie de parámetros. La imagen es convertida a blanco y negro y a formato RGB (por defecto la imagen es capturada en BGR) y la copia en blanco y negro es pasada al algoritmo de detección de caras que la procesa y devuelve por cada cara detectada cuatro valores que representan lo siguiente:

- **x**: posición horizontal de la esquina inferior izquierda del recuadro que enmarca la cara detectada.
- **y**: posición vertical de la esquina inferior izquierda del recuadro que enmarca la cara detectada.

- **w**: número de píxeles de anchura del marco.
- **h**: número de píxeles de altura del marco.

Para cada cara detectada además sería deseable hacer uso del algoritmo de reconocimiento de caras que devuelve la codificación de cada cara detectada pero como se ha explicado anteriormente este algoritmo es lo suficientemente lento como para impedir un correcto procesamiento en tiempo real por lo que se define el parámetro `ENCODING_FREQ` que nos va a decir con qué frecuencia vamos a efectuar dicha codificación. Por defecto el valor de dicho parámetro es 10 lo que significa que efectuamos dicho procesamiento en uno de cada diez frames capturados. La excepción a este valor se da cuando el número de caras detectadas es diferente a la iteración anterior en cuyo caso omitimos esta regla y procedemos a obtener la codificación de las caras detectadas. Además, la lista devuelta se itera ordenada por el valor `x` lo que nos va a hacer que no tengamos que recalcular la codificación de cada cara cuando hay más de una asumiendo para presentar las etiquetas correspondientes en las caras que éstas están en el mismo orden horizontal que la última vez que se analizaron.

El código correspondiente a este paso es el siguiente:

```
predictions = []
encoding_count += 1
last_faces_count = len(last_faces)
last_faces = []
_, frame = video_capture.read()

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
faces = model_helper.face_detection.detectMultiScale(gray, 1.3, 5)

do_encode = encoding_count >= ENCODING_FREQ or last_faces_count != len(faces)

if (do_encode):
    face_encodings = []

    face_index = 0
```

4.2.5.3 Procesamiento de caras

A continuación, se va a proceder a explicar el paso de procesamiento de caras que se ejecuta por cada una de las caras detectadas, el diagrama de flujo es el siguiente:

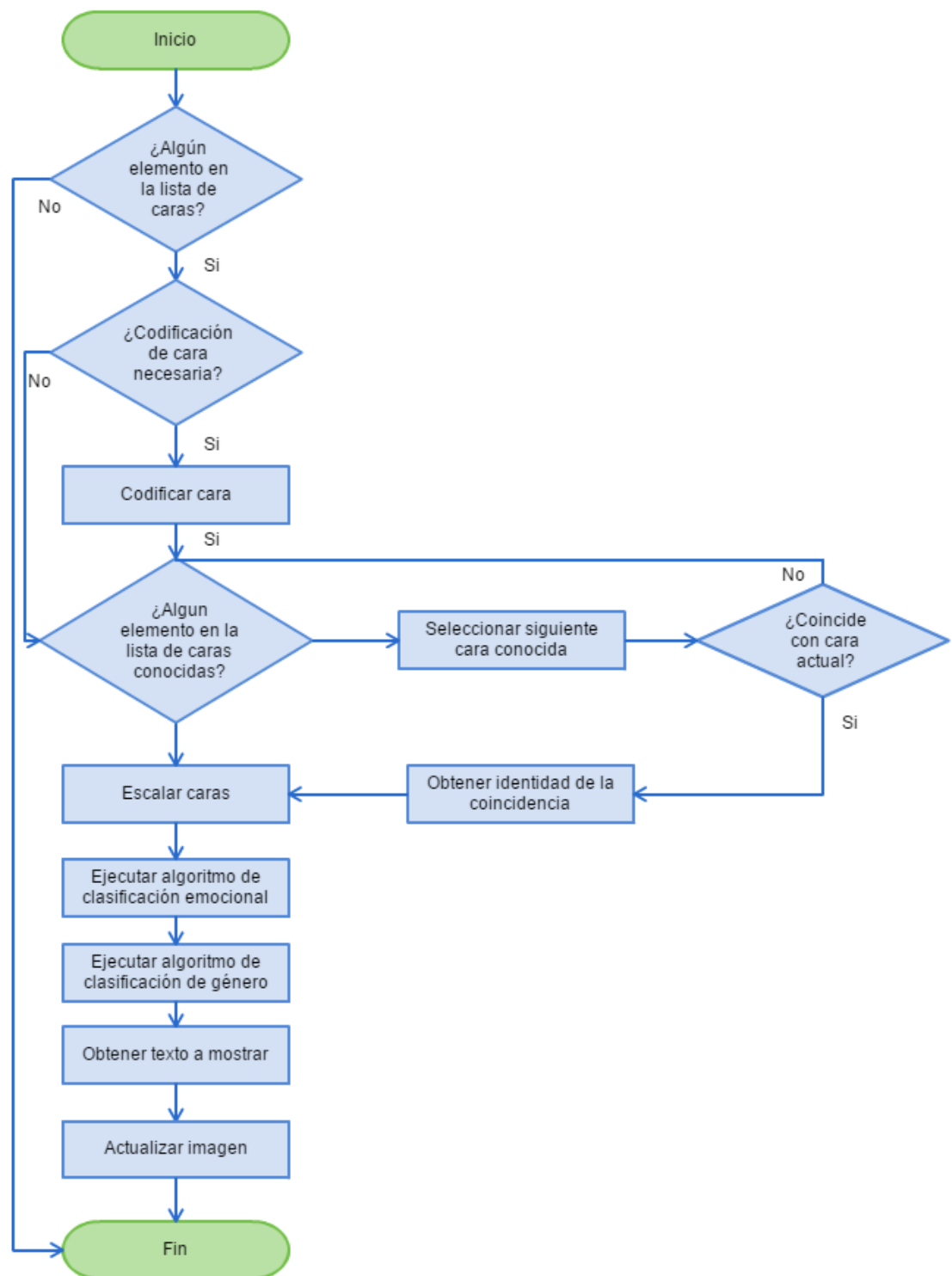


Figura 34: Diagrama de flujo del procesamiento de caras

Se itera sobre la lista de caras detectadas y se comprueba si debe codificarse para obtener la lista de identificadores faciales. El valor de variable de comprobación se ha calculado

anteriormente, concretamente en el paso anterior de detección de caras. En caso afirmativo se hace uso del modelo de reconocimiento de caras convirtiendo los valores x , y , w , h que como se ha explicado representa la posición de la esquina inferior izquierda del marco que representa la cara detectada junto con la anchura y altura de dicho marco a un formato en el que se representa la posición de las esquinas opuestas de dicho marco.

En caso de no realizarse una codificación se utilizará los valores de la última codificación realizada siendo esta por configuración como mucho la codificación de 10 frames anteriores

Independientemente de si la cara de esta iteración debe ser codificada o no el siguiente paso es comprobar si la presente (o más reciente) codificación realizada coincide con las codificaciones de las caras existentes en el sistema. El proceso de almacenamiento de nuevas identidades se explicará más adelante. En caso de coincidir se almacenará el valor de dicha identidad para ser utilizado posteriormente. Concretamente en caso de haber coincidencia, el valor de dicha identidad sustituirá el valor del género en la salida del sistema. Es decir, si por ejemplo detectara que en la imagen hay un hombre y que su estado emocional es alegre y la codificación de la cara de dicha persona no coincidiera con ninguna almacenada en el sistema la salida sería: “hombre contento”. Sin embargo, si hubiera una coincidencia y por ejemplo la identidad de dicha coincidencia fuera “Pedro” la salida mostraría: “Pedro contento”. En el siguiente ejemplo se ilustra este comportamiento.

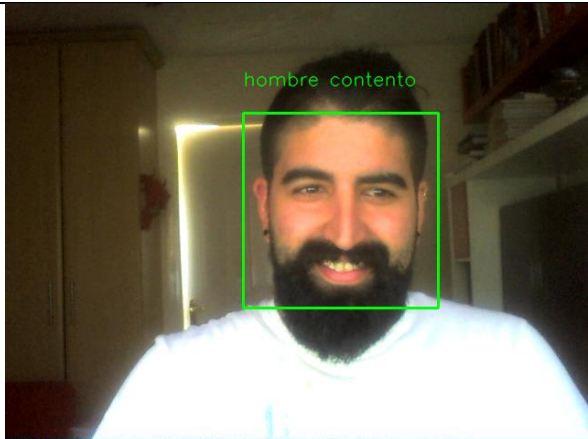
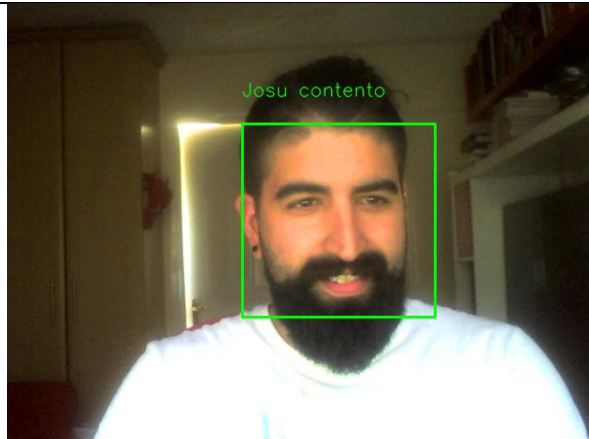
No hay coincidencia	Hay coincidencia
	

Tabla 7: Resultados del análisis de la imagen

El siguiente paso consiste en escalar la imagen de la cara procesada en la iteración al mismo tamaño que se utilizó en el entrenamiento de los algoritmos de clasificación de emoción y género, es decir, a 48x48 píxeles para posteriormente hacer uso de dichos modelos y obtener un resultado de clasificación en sendas categorías.

A continuación, con los resultados del análisis obtenido se obtendrá el resultado del texto final a mostrar en la imagen. Este proceso se describirá en detalle más adelante, sólo tener en cuenta que dependerá del idioma activo en el sistema.

Y ya para finalizar se actualizará la imagen para mostrar un marco que encuadre la cara detectada. Además, se añadirá el texto obtenido del análisis anterior y se aplicará un color a dicho marco: rojo para detección de género femenino y verde para masculino.

El código para todos los pasos de este apartado es el siguiente:

```
for (x,y,w,h) in sorted(faces, key=itemgetter(0)):
    pred_dict = label_dict.copy();
    face_index +=1
    face = frame[(y - y_offset):(y + h + y_offset),
                 (x - x_offset):(x + w + x_offset)]
    if (do_encode):
        print('re-encoding')
        face_encodings.append(face_recognition.face_encodings(frame,
[tuple([int(y), int(x+w), int(y+h), int(x)])])[0])
        encoding_count = 0

    try:
        if (len(face_encodings)>0 & face_index -1 < len(face_encodings)):
            for i in range(len(model_helper.known_faces)):
                match =
face_recognition.compare_faces([model_helper.known_faces[i][2]],
face_encodings[face_index-1])
                if match[0]:
                    pred_dict['NAME'] = model_helper.known_faces[i][0]
                    pred_dict['FULL_NAME'] = model_helper.known_faces[i][1]
                    break;

    except Exception as e:
        continue
    last_faces.append(cv2.cvtColor(face.copy(), cv2.COLOR_RGB2BGR))

    gray_face = gray[(y - y_offset_emotion):(y + h + y_offset_emotion),
                    (x - x_offset_emotion):(x + w + x_offset_emotion)]
    try:
        face = cv2.resize(face, (48, 48))
        gray_face = cv2.resize(gray_face, (48, 48))
    except:
```



```

        continue
    face = np.expand_dims(face, 0)
    face = preprocess_input(face)
    gender_label_arg = np.argmax(model_helper.gender_classifier.predict(face))
    gender = gender_labels[gender_label_arg]
    gender_label_window.append(gender)

    gray_face = preprocess_input(gray_face)
    gray_face = np.expand_dims(gray_face, 0)
    gray_face = np.expand_dims(gray_face, -1)
    emotion_label_arg =
np.argmax(model_helper.emotion_classifier.predict(gray_face))
    emotion = emotion_labels[emotion_label_arg]
    emotion_label_window.append(emotion)

    if len(gender_label_window) >= frame_window:
        emotion_label_window.pop(0)
        gender_label_window.pop(0)
    try:
        emotion_mode = mode(emotion_label_window)
        gender_mode = mode(gender_label_window)
    except:
        continue
    if gender_mode == gender_labels[0]:
        gender_color = (255, 0, 0)
    else:
        gender_color = (0, 255, 0)

    pred_dict['EMOTION'] = emotion_mode
    pred_dict['GENDER'] = gender_mode

    display_text = lang_helper.get_formatted_language_text(pred_dict)

    cv2.rectangle(frame, (x, y), (x + w, y + h), gender_color, 2)
    cv2.putText(frame, display_text, (x, y - 30), font,
                .7, gender_color, 1, cv2.LINE_AA)

    predictions.append(pred_dict)

```

4.2.5.4 Opciones del sistema

Después de procesar cada imagen en cada iteración es cuando se comprueba si el usuario ha realizado alguna acción sobre el sistema que modifique el flujo de trabajo. El dispositivo de entrada utilizado es el teclado si bien pulsar la tecla espacio sirve para activar la entrada por micrófono. Se detallan en los siguientes apartados las opciones principales del sistema, seguidamente se especifica el contenido de los archivos de audio utilizados en el menú auditivo. A continuación, se mostrará el diagrama de flujo de los menús y para finalizar se mostrará parte del código para ejecutar estas funcionalidades.

Se describe en esta tabla las opciones principales del menú del sistema. Además, en el Anexo 7.2 se incluye el *script* que transforma el texto en audio y lo guarda en la ubicación correspondiente al idioma utilizado.

Opción	Teclado	Micrófono (es)	Micrófono (en)
Activar entrada de micrófono	Espacio	N/A	N/A
Cambiar idioma	L	Idioma	Language
Describir las personas de la imagen de entrada	A	Quien	Who
Guardar la identidad de la persona detectada en la imagen	S	Guardar	Save
Cancelar	N/A	Cancelar	Cancel
Salir del sistema	Q	Salir	Quit

Tabla 8: Comandos de entrada

4.2.5.4.1 Audio de las opciones

En la tabla siguiente se muestran un listado de los diálogos que el sistema utiliza para comunicarse con el usuario ordenados alfabéticamente por clave.

Clave	Español	Inglés
canceled	Cancelado	Canceled
choose	Seleccione una opción. O diga: Comandos. Para oír una lista de comandos sonoros. O: Teclas. Para oír una lista de comandos de entrada.	Select an option. Or say Options to hear a list of available commands. Or say Keys to hear a list of available keys
choose_short	Seleccione una opción	Select an option

commands	Diga: ¿Quién? Para obtener una descripción de las personas en la imagen. Diga: ¿Qué? Para obtener una descripción general de la imagen. Diga: Guardar. Para guardar en el sistema el nombre de la persona en la imagen. Diga: Idioma. Para cambiar el idioma al siguiente disponible. Diga: Cancelar. Para continuar o diga: Repetir. Para repetir las opciones.	Say: Who. To get a description of the people in the image. Say: What. To get a general description of the image. Say: Save. To save the name of the person in the image. Say: Language. To change the language. Say: Cancel. To continue. Say: Repeat. To repeat the list of available options
keys	Pulsa la tecla: A. Para obtener una descripción de las personas en la imagen. Pulsa la tecla: Z. Para obtener una descripción general de la imagen. Pulsa la tecla: S. Para guardar en el sistema el nombre de la persona en la imagen. Pulsa la tecla: L. Para cambiar el idioma al siguiente disponible. Pulsa la tecla: Q. Para cancelar o Pulsa la tecla: R. Para repetir las opciones	Press "A". To get a description of the people in the image. Press "Z". To get a general description of the image. Press "S". To save the name of the person in the image. Press "L". To change the language. Press "Q". To continue. Press "R". To repeat the list of available options
lang_change	Idioma cambiado a español	Language has been changed to english
more_than_one_face	Se ha detectado más de una persona, inténtelo de nuevo con una persona sólo por favor	There appears to be more than one person, try again with one person only please
no_image	Lo siento, no soy capaz de describir la imagen	Sorry, I cannot understand what's in the image
not_understand	Lo siento, no he entendido bien	Sorry, I did not catch that
ok	De acuerdo	OK
one_moment	Un momento por favor	One moment please

repeat_options	¿Quieres que repita las opciones disponibles?	Do you want me to repeat the available options?
saved	Ha sido guardado correctamente	Has been saved correctly
saving	Guardando	Saving
sorry_understand	Lo siento, no te he entendido	Sorry, I did not get that
who	Diga el nombre de la persona detectada o cancelar después del pitido por favor	Say the name of the person detected or cancel after the beep please

Tabla 9: Audios de la interacción

Además de los audios de los menús existen otros audios pregrabados que se utilizan para describir el análisis de la imagen. Un problema detectado en la implementación de estos audios descriptivos es que no es tan sencillo como los anteriores donde se ha grabado la versión en español e inglés únicamente. A la hora de describir a la persona detectada influye el género. En inglés no hay problema a la hora de representar esto, pero si en otros idiomas como el español. Esto ha motivado que tenga que idear un método para representar la información analizada tanto visualmente como auditivamente teniendo en cuenta no solo el idioma seleccionado sino también el género. Por ejemplo, para el caso de que el sistema detecte un hombre y una mujer contentos. En inglés esto se representará como “happy man” y “happy woman” donde lo único que cambia entre uno y otro es la palabra que describe el género de la persona. Sin embargo, en español esto se traduce como “hombre contento” y “mujer contenta” donde no sólo cambia la palabra que designa el género sino también la terminación de la palabra que representa la emoción. En el apartado de sistema de soporte multi-lenguaje se detalla cómo se ha solventado este problema.

Los archivos de audio generados para esto son los siguientes:

Clave	Español (masculino)	Español (femenino)	Inglés
man	Hombre	Hombre	Man
woman	Mujer	Mujer	Woman
angry	Enfadado	Enfadada	Angry
disgust	Asqueado	Asqueada	Disgusted
happy	Contento	Contenta	Happy
neutral	Neutral	Neutral	Neutral
sad	Triste	Triste	Sad
surprise	Sorprendido	Sorprendida	Surprised

Tabla 10: Audios del análisis

4.2.5.4.2 Diagrama de flujo del menú de opciones

En cada iteración después de analizar la imagen de entrada se comprueba si hay alguna interacción con el usuario. Esta interacción es iniciada siempre por teclado bien para activar el menú auditivo y capturar las instrucciones de forma verbal o bien para acceder directamente a las opciones por teclado. En el siguiente diagrama se muestra el diagrama de flujo que explica el funcionamiento de esto para al final obtener una de las actualmente cinco opciones principales de menú o nada en caso de no haber interacción

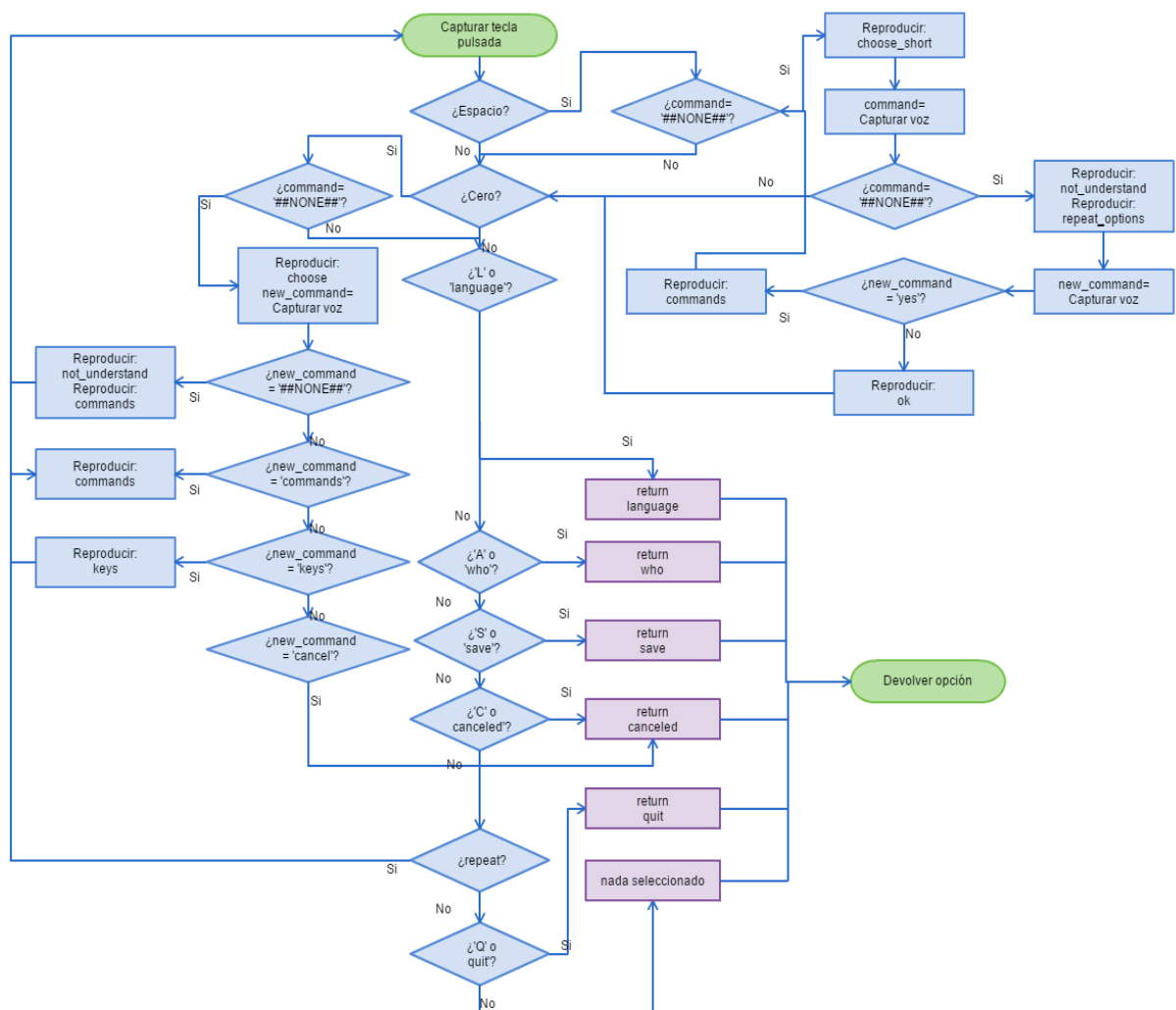


Figura 35: Diagrama de flujo del menú de opciones

Después del proceso de captura el sistema sabrá que acción ha seleccionado el usuario en caso de haber seleccionado alguna. Las opciones disponibles son:

Opción	Descripción
language	Cambia el idioma del sistema al siguiente disponible
who	Reproduce de forma sonora la descripción de las personas detectadas en la imagen en caso de haber detectado alguna
save	Guarda la identidad de la persona detectada en la imagen en caso de haberse detectado alguna
canceled	Cancela la interacción con el usuario
quit	Finaliza la captura de imágenes y cierra la aplicación

Tabla 11: Descripción de las opciones

A continuación, se incluye el código representado en el diagrama de flujo y en los siguientes apartados se detallan las acciones realizadas para cada una de las posibles opciones del sistema.

```
def get_command(self, c):
    command = '##NONE##'
    if (c==' '):
        try:
            while command == '##NONE##':
                self.talk('choose_short')
                nc = chr(cv2.waitKey(2) & 255)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                command = self.capture_selected_command()
                if (command=='##NONE##'):
                    self.talk('not_understand')
                    self.talk('repeat_options')
                    new_command = '##NONE##'
                    new_command = self.capture_custom_command(['yes',
'no'])

                    if (new_command == 'yes'):
                        self.talk('commands')
                    else:
                        self.talk('ok')
                        command = 'cancel'
                        break;

                else:
                    break
        except Exception as e:
```

```

        print('*c*****')
        print(e)
        print('*c*****')
    elif (c=='0'):
        try:
            while command == '##NONE##':
                self.talk('choose')
                nc = chr(cv2.waitKey(2) & 255)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                new_command = '##NONE##'
                new_command = self.capture_custom_command(['commands',
'keys', 'cancel']))

            if (new_command=='##NONE##'):
                self.talk('not_understand')
                self.talk('commands')
                return self.get_command(' ')
            elif (new_command=='commands'):
                self.talk('commands')
                return self.get_command(' ')
            elif (new_command=='keys'):
                self.talk('keys')
                return self.get_command(' ')
            elif (new_command=='cancel'):
                return 'cancel'
            else :
                self.talk('not_understand')
                self.talk('commands')
                return self.get_command(' ')
        except Exception as e:
            print('*c*****')
            print(e)
            print('*c*****')

    if (c=='L' or command=='language'):
        return 'language'
    elif (c=='A' or command=='who'):
        return 'who'
    elif (c=='S' or command=='save'):
        return 'save'
    elif (c=='C' or command=='cancel'):
        self.talk('canceled')
        return 'cancel'

```

```
elif (command=='repeat'):
    self.talk('commands')
    return self.get_command(' ')
if (c=='Q' or command=='quit'):
    return 'quit'
```

4.2.5.4.3 Opción LANGUAGE

Al seleccionar esta opción se cambia el idioma al siguiente disponible. Actualmente el sistema da soporte a dos idiomas: español e inglés y por defecto la herramienta arranca en español. En el apartado de soporte multi-lenguaje se detalla el funcionamiento para el manejo de múltiples idiomas, sólo destacar que el sistema soportaría fácilmente la implementación de más idiomas.

Para cambiar el idioma al siguiente disponible se llama a la función *switch_to_next_language* que obtiene la información de configuración del siguiente idioma y actualiza los parámetros de idioma de la clase *Lang_Helper* encargada del tratamiento de los idiomas.

```
def change_language(self, new_lang):
    self.config_audios = []
    with open(self.language_path+new_lang+'/audio_config.txt') as f:
        for line in f:
            self.config_audios.append(line.rstrip())

    l = gettext.translation('text_', localedir=self.language_path,
languages=[new_lang])
    l.install()
    self._ = l.gettext
    self.current_language = new_lang

def switch_to_next_language(self):
    idx = self.available_languages.index(self.current_language)
    new_idx = 0
    if (idx+1 < len(self.available_languages)):
        new_idx = idx+1
    self.change_language(self.available_languages[new_idx])
```

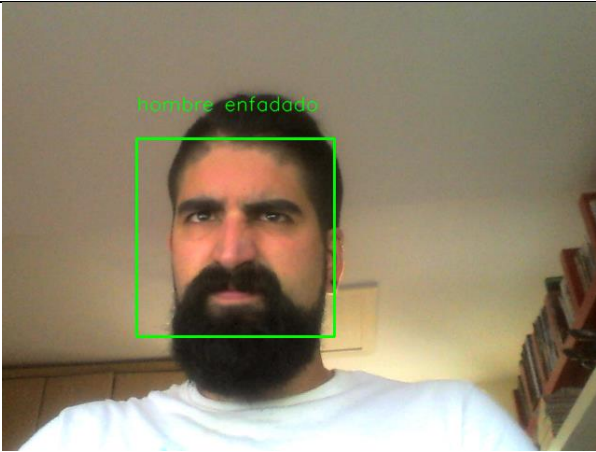
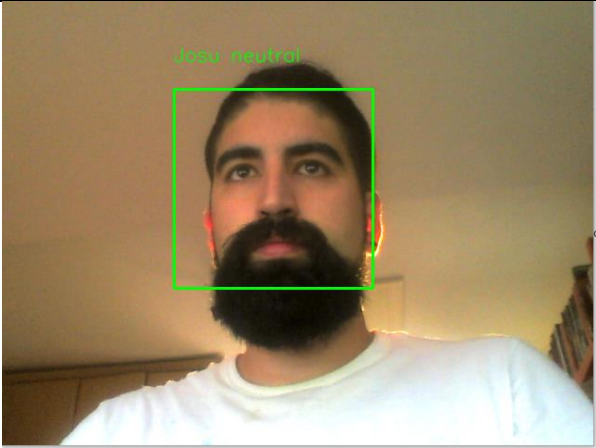
4.2.5.4.4 Opción WHO

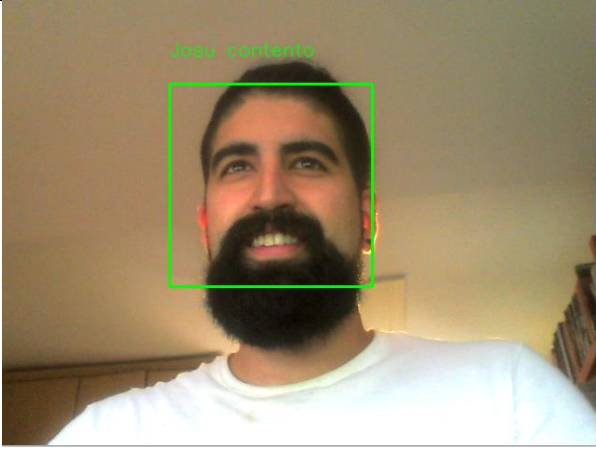
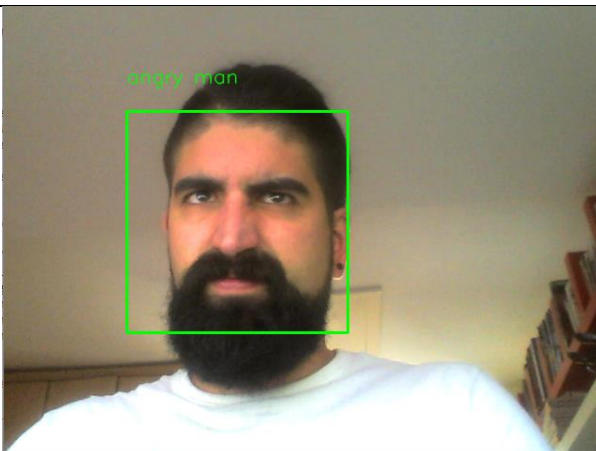
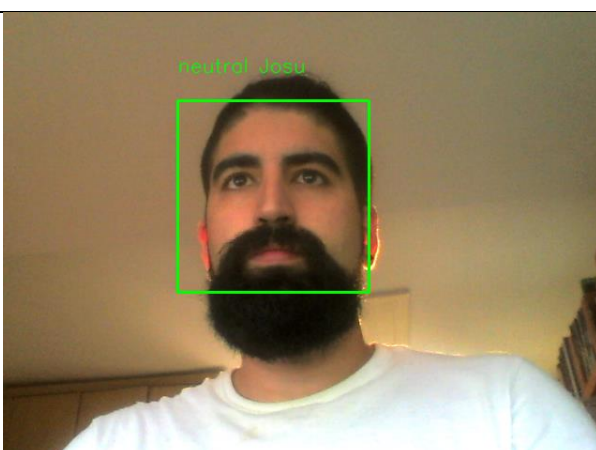
La opción WHO sirve para obtener una descripción auditiva del número de caras detectadas en la imagen, así como de su género y su estado emocional. En el caso de que la identidad

de alguna de las caras detectadas corresponda a alguna de las almacenadas en el sistema se reproducirá el nombre de la persona guardada en vez de su género.

Hay que apuntar la información del análisis se encuentra en todo momento disponible de forma visual, esta opción simplemente se limita a reproducir auditivamente la información capturada en el momento que se activa esta opción.

La descripción auditiva se realizará en el idioma seleccionado en ese momento en el sistema. Veamos unos ejemplos:

Imagen después del análisis	Características	Transcripción del audio reproducido
	Idioma: español Identidad: no existente en el sistema	Hombre enfadado
	Idioma: español Identidad: coincidencia	Josu neutral

	Idioma: español Identidad: coincidencia	Josu contento
	Idioma: inglés Identidad: coincidencia	Angry man
	Idioma: inglés Identidad: coincidencia	Neutral Josu

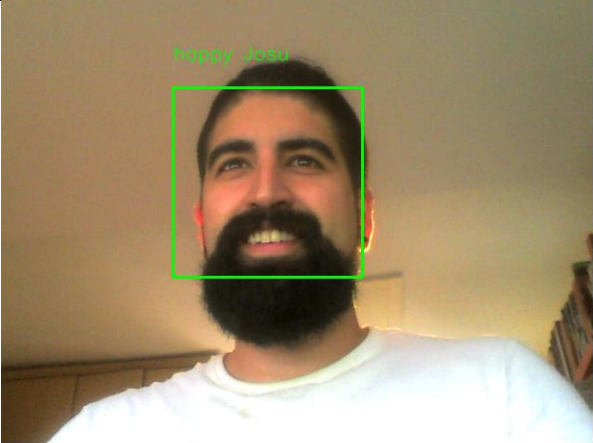
	Idioma: inglés Identidad: coincidencia	Happy Josu
---	--	------------

Tabla 12: Ejemplos de análisis

Para que se pueda realizar esta opción el sistema debe haber sido capaz de detectar por lo menos una cara. De no ser así se reproduce una grabación explicando que el análisis no ha detectado nada. Si hay más de una cara se reproducirá la información de cada una de ellas.

A continuación, se muestra el código de esta opción donde el método `get_formatted_language_audios` recibe como parámetro un objeto que contiene la información de todas las caras detectadas y devuelve una lista de archivos de audio a reproducir.

```
if (len(predictions) > 0):
    lang_audios = lang_helper.get_formatted_language_audios(predictions)
    for lang_audio in lang_audios:
        lang_helper.play(lang_audio)
else:
    lang_helper.talk('no_image')
```

4.2.5.4.5 Opción SAVE

La opción SAVE o guardar sirve para almacenar en el sistema la identidad de la persona detectada en la imagen para ser utilizada posteriormente. Para poder guardar esta información en análisis realizado en la imagen debe haber sido capaz de detectar una y solo una cara. No se permite guardar más de una identidad a la vez.

El código encargado de realizar esta acción es el siguiente:

```
print('*** Save person selected *** ')
try:
    if (len(last_faces)==1):
```

```

        name = '##NONE##'
        while name == '##NONE##':
            lang_helper.talk('who')
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
            else:
                name = lang_helper.capture_name()
                if (name=='##NONE##'):
                    lang_helper.talk('not_understand')
                elif (name == 'cancel'):
                    lang_helper.talk('canceled')
                    break
                else:
                    print('saving face...')
                    full_name = model_helper.save_face(name,
lang_helper.current_language, last_faces[0], face_encodings[face_index-1])
                    print('/////////')
                    print(full_name)
                    print(lang_helper.audio_path + 'known/' + full_name +
'.mp3')

                    if (full_name!=''):
                        lang_helper.play(lang_helper.audio_path + 'known/'
+ full_name + '.mp3')

                        lang_helper.talk('saved')
                        break
                elif (len(last_faces)>1):
                    lang_helper.talk('more_than_one_face')
                else:
                    lang_helper.talk('no_image')
        except:
            continue

```

Y en el siguiente diagrama de flujo se puede apreciar la lógica del proceso.

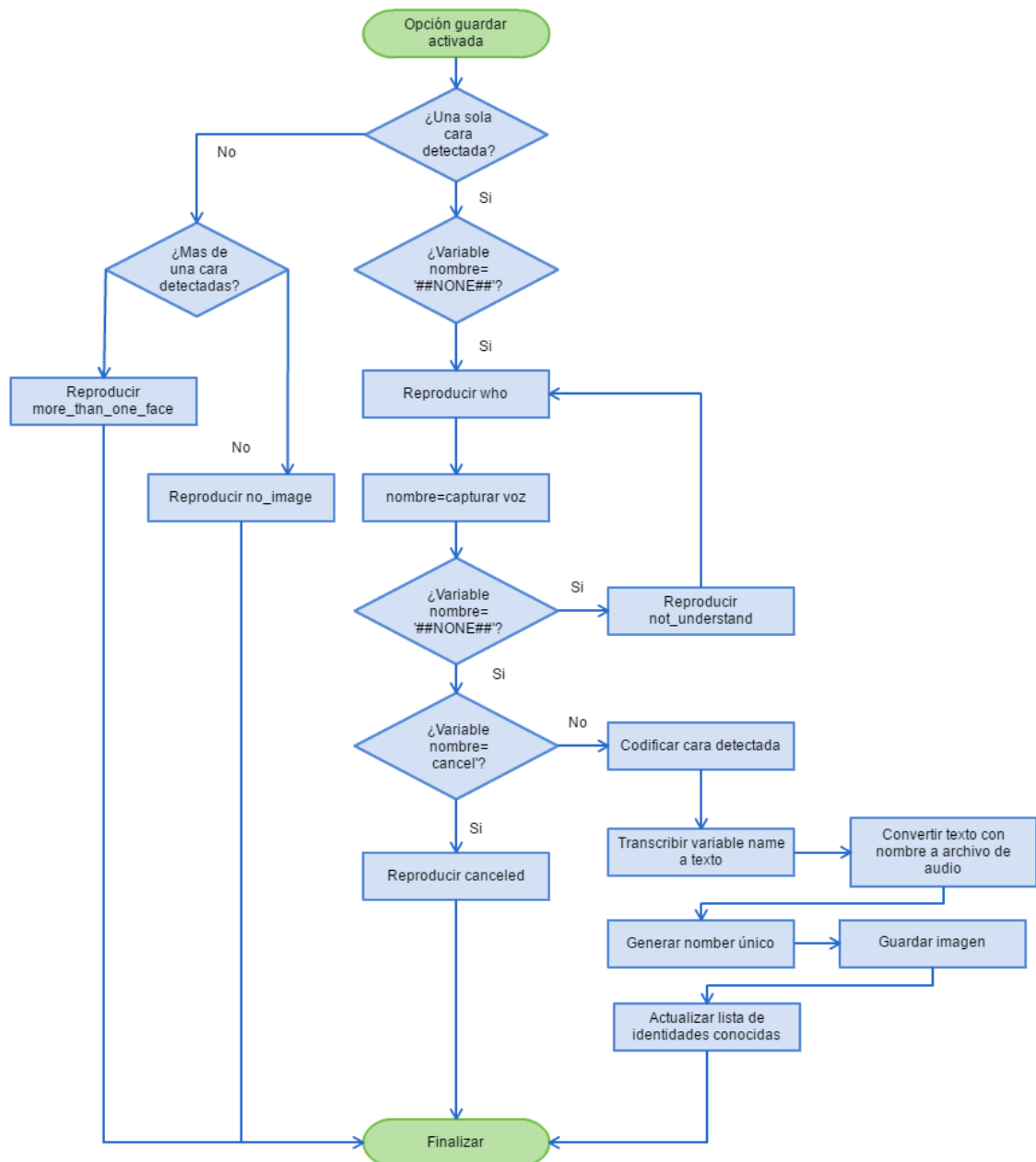


Figura 36: Diagrama de flujo de la opción guardar

En caso satisfactorio se guardan en el sistema dos archivos, la imagen de la cara capturada y un archivo de audio generado a partir de la transcripción de la grabación realizada con el nombre de la persona detectada. La cara detectada además es codificada y agregada a la lista que contiene la codificación de todas las imágenes guardadas en el sistema y que se carga al inicio de la aplicación.

La ubicación de los archivos de imágenes guardados es:

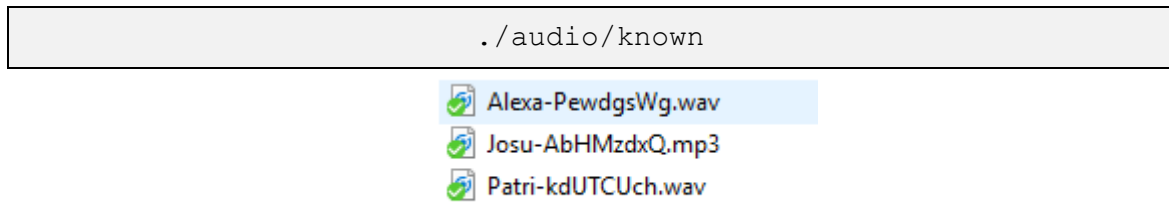


Figura 37: Audios de identidades guardadas en el sistema

Y la ubicación de los archivos de audio generados es:



Figura 38: Imágenes de identidades guardadas en el sistema

4.2.5.4.6 Opción CANCEL

La opción cancelar se utiliza para cancelar la interacción con el sistema tanto en el menú principal como cuando se ha decidido guardar la identidad de la cara detectada.

4.2.5.4.7 Opción QUIT

La opción QUIT o salir se utiliza para finalizar la captura de imágenes y salir de la aplicación.

4.2.6 Sistema de soporte multi-lenguaje

Desde el inicio del desarrollo de la aplicación se decidió dotarla de flexibilidad suficiente para poder manejarse en distintos idiomas. Pero un problema añadido fue el tener que diferenciar dentro de un mismo idioma (concretamente el español) el género de los adjetivos utilizados en la descripción del análisis realizado en la imagen. Un ejemplo de este problema se explica en el apartado donde se describe el procesamiento de las caras. Además, el hecho de tratar los resultados del análisis por separado crea otro problema a la hora de saber el orden correcto de las palabras para realizar la descripción.

Voy primeramente a explicar cómo se ha desarrollado el soporte de varios idiomas y después mostraré como como se han solventado los problemas del género y el orden de las palabras.

Para dotar a la herramienta de soporte multilinguaje se ha utilizado la librería *gettext* (Peter Funk, 2017).

En la página (Sweigart, 2014) hay un tutorial bastante completo del proceso seguido pero voy a relatar aquí los pasos principales.

Primeramente, se ha generado un archivo *.pot* a partir de un archivo *.py* utilizando el script *pygettext.py*. A continuación, se muestra una parte del archivo generado:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR ORGANIZATION
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: \n"
"POT-Creation-Date: 2017-06-09 08:29+0100\n"
"PO-Revision-Date: 2017-06-15 16:13+0100\n"
"Language-Team: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Generated-By: pygettext.py 1.5\n"
"X-Generator: Poedit 2.0.2\n"
"Last-Translator: \n"
"Plural-Forms: nplurals=2; plural=(n != 1);\n"
"Language: en\n"

#: texts.py:1
msgid "man"
msgstr "man"

#: texts.py:2
msgid "woman"
msgstr "woman"

#: texts.py:3
msgid "angry"
msgstr "angry"

#: texts.py:4
msgid "disgust"
```

```
msgstr "disgusted"

#: texts.py:5
msgid "happy"
msgstr "happy"
```

No es necesario tener todo el texto inicialmente ya que se puede modificar posteriormente.

Una vez que se tiene el archivo *.pot* se ha utilizado la aplicación Poedit (Poedit, 2017) donde se cargará y se especificará la traducción correspondiente para después guardarlo lo que generará dos archivos: un *.po* con la traducción y un *.mo* que es el que utilizará posteriormente *gettext*. Se debe generar un par de estos archivos por cada idioma y guardarse en una subcarpeta llamada LC_MESSAGES para cada idioma.

En nuestro caso al tener dos idiomas nuestra jerarquía de carpetas y archivos es la siguiente:

```
lang
├── es
│   ├── LC_MESSAGES
│   │   ├── text_.po
│   │   └── text_.mo
│   └── en
│       ├── LC_MESSAGES
│       │   ├── text_.po
│       │   └── text_.mo
```

Una vez en disposición de los archivos con las traducciones generados es bastante sencillo obtener la traducción correspondiente como se puede observar en el siguiente ejemplo donde primeramente se carga el idioma español y se imprime el valor correspondiente al identificador "happy" para después hacer lo mismo con el idioma inglés obteniéndose como resultado "contento" y "happy".

```
In [6]: import gettext

es = gettext.translation('text_', localedir='./lang/', languages=['es'])
es.install()
_es = es.gettext
print(_('happy'))

en = gettext.translation('text_', localedir='./lang/', languages=['en'])
en.install()
_en = en.gettext
print(_('happy'))

contento
happy
```

Figura 39: Ejemplo multi-lenguaje

Con todo esto nuestro sistema ya es capaz de comunicarse en varios idiomas. Es ahora el momento de explicar el proceso para solventar el problema de la existencia en algunos idiomas como el español de distintas palabras para definir el mismo adjetivo, pero aplicado a distinto género y también el problema de representar en el orden correcto el adjetivo y el sustantivo utilizado en la descripción del análisis de la imagen.

Primeramente, se ha utilizado el mismo sistema anterior para generar varios idiomas, pero esta vez aplicado a un mismo idioma. Para ello se ha generado un nuevo par de archivos para las palabras en género masculino y otro par para el femenino, pero únicamente para el idioma español.

A continuación, se puede observar parte del contenido de ambos archivos. Sólo apuntar que estos archivos solo contienen texto que va a ser utilizado para catalogar el contenido de una imagen. El resto del texto utilizado por la herramienta se encuentra en los archivos de idioma específicos de cada idioma y explicados anteriormente.

text_man.po	text_woman.po
#: texts.py:1 msgid "man" msgstr "hombre"	#: texts.py:1 msgid "man" msgstr "hombre"
#: texts.py:2 msgid "woman" msgstr "mujer"	#: texts.py:2 msgid "woman" msgstr "mujer"
#: texts.py:3 msgid "angry" msgstr "enfadado"	#: texts.py:3 msgid "angry" msgstr "enfadada"
#: texts.py:4 msgid "disgust" msgstr "asqueado"	#: texts.py:4 msgid "disgust" msgstr "asqueada"
#: texts.py:5 msgid "happy" msgstr "contento"	#: texts.py:5 msgid "happy"msgstr "sorprendida" msgstr "contenta"
#: texts.py:6 msgid "neutral" msgstr "neutral"	#: texts.py:6 msgid "neutral" msgstr "neutral"

#: texts.py:7 msgid "sad" msgstr "triste"	#: texts.py:7 msgid "sad" msgstr "triste"
#: texts.py:8 msgid "surprise" msgstr "sorprendido"	#: texts.py:8 msgid "surprise"

Tabla 13: Archivos de idioma por género

Estos archivos adicionales junto con las versiones *.mo* se guardan en la misma carpeta del idioma correspondiente. En nuestro caso al ser sólo necesarios para el idioma español la jerarquía resultante sería la siguiente:

```
lang
├── es
│   └── LC_MESSAGES
│       ├── text_.po
│       ├── text_.mo
│       ├── text_man.po
│       ├── text_man.mo
│       ├── text_woman.po
│       └── text_woman.mo
```

Y de esta manera ya dispondríamos de todas las traducciones necesarias independientemente del idioma y del género dentro del mismo idioma, pero eso no es todo. Hasta ahora sólo nos hemos preocupado de las traducciones de texto, pero teniendo en cuenta el propósito de la herramienta también necesitamos traducciones de audio. Anteriormente se ha explicado cómo generar archivos de audio en distintos idiomas a partir de texto, ahora sólo voy a explicar donde se encuentran guardados dichos archivos para posteriormente explicar el sistema ideado para hacer uso de ellos.

```

lang
├── es
│   ├── emotion
│   │   ├── man
│   │   │   ├── angry.mp3 > grabación: Enfadado.
│   │   │   ├── disgust.mp3 > grabación: Asqueado.
│   │   │   ├── happy.mp3 > grabación: Contento.
│   │   │   ├── neutral.mp3 > grabación: Neutral.
│   │   │   ├── sad.mp3 > grabación: Triste.
│   │   │   └── surprise.mp3 > grabación: Sorprendido.
│   │   └── woman
│   │       ├── angry.mp3 > grabación: Enfadada.
│   │       ├── disgust.mp3 > grabación: Asqueada.
│   │       ├── happy.mp3 > grabación: Contenta.
│   │       ├── neutral.mp3 > grabación: Neutral.
│   │       ├── sad.mp3 > grabación: Triste.
│   │       └── surprise.mp3 > grabación: Sorprendida.
│   └── gender
│       ├── man.mp3 > grabación: Hombre.
│       └── woman.mp3 > grabación: Mujer.
└── en
    ├── emotion
    │   ├── angry.mp3 > grabación: Angry.
    │   ├── disgust.mp3 > grabación: Disgusted.
    │   ├── happy.mp3 > grabación: Happy.
    │   ├── neutral.mp3 > grabación: Neutral.
    │   ├── sad.mp3 > grabación: Sad.
    │   └── surprise.mp3 > grabación: Surprised.
    └── gender
        ├── man.mp3 > grabación: Man.
        └── woman.mp3 > grabación: Woman.

```

Además de las carpetas y archivos mencionados dentro de cada carpeta de idioma se incluye otra carpeta llamada *speech* que contiene todos los archivos de audio utilizados en la interacción con el usuario excepto los específicos para la descripción de la imagen que se encuentran en las carpetas *gender* y *emotion* y aquellos archivos de audio que contienen la identidad de alguna persona guardada en el sistema que se encuentra en la carpeta:

```
./audios/known
```

Ahora que ya tenemos todo el texto y audio de cada idioma organizado necesitamos un sistema para poder hacer uso de ellos. Para ello se ha ideado lo siguiente: haciendo uso de unos archivos de configuración que especifique que archivo de traducción o de audio utilizar en cada caso concreto. Explicaré primero la configuración de texto.

Dentro de la carpeta de cada idioma correspondiente se ha creado un archivo llamado *text_config.txt* el cual contiene una sola línea que va a definir el orden de las palabras a mostrar y como deben buscarse. Por ejemplo, en inglés el archivo *text_config.txt* contiene lo siguiente:

```
: [EMOTION] [GENDER]
```

Donde los valores en corchete serán reemplazados con los valores que el sistema haya detectado en el orden en que se encuentran en el archivo. Por ejemplo, si el sistema ha detectado una mujer con estado emocional contento reemplazará [EMOTION] con “happy” y [GENDER] con “woman” resultando en “happy woman”. Los dos puntos del principio del archivo de traducción sirven para especificar subcarpetas en el caso de que el género determine un valor distinto para el adjetivo. Como en inglés este problema no ocurre no tiene ningún efecto. Pero ahora vamos a ver el archivo de configuración de texto para el idioma español:

```
[GENDER] : [GENDER] [EMOTION]
```

En este caso se observa que los valores después de los dos puntos tienen el orden inverso al inglés por lo que al reemplazar los valores del ejemplo anterior y aplicar el orden actual el resultado sería “mujer contenta” o “mujer contento”. Para identificar el resultado correcto se utiliza el valor anterior a los dos puntos que básicamente viene a decir que el género influye en el resultado y que se utilice el valor del género detectado para identificar el texto correcto. Como en nuestro ejemplo el género es femenino el sistema utilizará el archivo ***text_woman.po*** en vez de ***text_man.po***.

Veamos ahora el funcionamiento para el audio haciendo uso del archivo de configuración *audio_config.txt* de las carpetas de idioma. Este archivo contiene una línea por cada archivo de audio a reproducir y esta ordenado por orden de reproducción. En inglés el contenido es el siguiente:

```
EMOTION:emotion/[EMOTION].mp3  
GENDER:gender/[GENDER].mp3
```

Una vez más los valores en corchetes se reemplazarán por el valor detectado por el sistema. En este caso el valor antes de los dos puntos se utiliza para determinar el tipo de elemento y el valor de después es la ubicación del archivo de audio a reproducir. Por lo que en este caso

se obtiene la ubicación de los dos archivos en inglés que van a reproducirse para describir la imagen. Siguiendo con el ejemplo anterior el resultado sería:

```
lang/en/emotion/happy.mp3  
lang/en/gender/woman.mp3
```

Y al reproducir dichos archivos en orden se escucharía: “happy woman”.

En el caso del idioma español volvemos a tener el problema de la dependencia del género por lo que el archivo de configuración es el siguiente:

```
GENDER:gender/[GENDER].mp3  
EMOTION:emotion/[GENDER]/[EMOTION].mp3
```

Y como se ha explicado los valores en corchetes se reemplazan con los valores detectados por lo que el resultado es:

```
lang/es/gender/woman.mp3  
lang/es/emotion/woman/happy.mp3
```

Escuchándose: “mujer contenta”.

Y con esto se cubre la explicación sobre el funcionamiento del sistema multilenguaje. Sólo apuntar que en el caso de que la identidad de la persona detectada se corresponda con alguna almacenada en el sistema el audio y texto utilizados para el género se reemplazarán con el nombre de la persona identificada.

A continuación, se muestra el código de las funciones encargadas del proceso explicado en este apartado:

```
def get_formatted_language_audios(self, predictions):  
    lang_audios = []  
    try:  
        print(predictions)  
        for prediction in predictions:  
            for audio in self.config_audios:  
                key = audio.split(':')[0]  
                if (key == 'GENDER' and prediction['FULL_NAME'] != ''):
```

```

        audio_path = self.audio_path + 'known/' +
prediction['FULL_NAME'] + '.mp3'
        lang_audios.append(audio_path)
    else:
        audio_path = self.language_path + self.current_language +
        '/' + audio.split(':')[1]
        for key in prediction:
            audio_path = audio_path.replace('['+key+']',
prediction[key])
        lang_audios.append(audio_path)

except Exception as e:
    print('*a*****')
    print(e)
    print('*a*****')
return lang_audios

def get_formatted_language_text(self, prediction):
    lang_text = ''
    try:
        text_config = ''
        with open(self.language_path + self.current_language +
        '/text_config.txt') as f:
            for line in f:
                text_config += line.rstrip()
            g = text_config.split(':')[0]
            lang_text = text_config.split(':')[1]

            for key in prediction:
                g = g.replace('['+key+']', prediction[key])

            l = gettext.translation('text_' + g, localedir=self.language_path,
languages=[self.current_language])
            l.install()
            __ = l.gettext
            t = ''
            if (prediction['NAME'] != ''):
                t = prediction['NAME']
            else:
                if (prediction['GENDER'] != ''):
                    t = __ (str(prediction['GENDER']))

            lang_text = lang_text.replace('[GENDER]', t)

```

```
t = ''
if(prediction['EMOTION'] != ''):
    t = ____(prediction['EMOTION'])

lang_text = lang_text.replace('[EMOTION]', t)
except Exception as e:
    print('*t*****')
    print(e)
    print('*t*****')
return lang_text
```

4.3 Evaluación de la calidad y aplicabilidad del producto

Se considera que el producto final ha satisfecho los requerimientos iniciales. Los resultados del análisis de las imágenes de entrada son bastante precisos en las pruebas realizadas. El procesamiento se efectúa lo suficientemente rápido para considerarse tiempo real y la herramienta es estable.

Además, soporta varios idiomas y las pruebas en ellos han sido satisfactorias. Esto permite una adecuada comunicación con el usuario siendo el público destinado personas ciegas. Una vez la herramienta funcionando lo único necesario para realizar una interacción es pulsar la tecla espacio que inicia el menú auditivo y a través de él y sólo con la voz se puede interactuar con la herramienta y obtener el *feedback* del análisis realizado. Además del menú auditivo la herramienta muestra de forma visual en todo momento los resultados del análisis y permite el acceso a las opciones del sistema a través del teclado facilitando su usabilidad.

En cuanto a la calidad del análisis de la imagen en los apartados anteriores se ha mostrado el porcentaje de precisión de los modelos. Si bien el porcentaje para el análisis de emociones no es muy alto los trabajos futuros se pretende mejorarlo.

En resumen, las pruebas realizadas con la herramienta determinan que esta cumple los objetivos de ser capaz de analizar una imagen de entrada y obtener información lo suficiente precisa, así como permitir una interacción a través de comandos vocales.

CAPÍTULO 5

CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Resumen del problema

El problema inicial es la dificultad de las personas ciegas para asimilar información presentada en formato visual. Concretamente ser capaces de identificar la identidad, el estado de ánimo y el género de la persona o personas con las que está interactuando.

Para darle una posible solución a este problema se ha construido una herramienta que analiza imágenes de entrada a través de la webcam para detectar posibles caras y clasificar éstas entre una lista de posibles estados de ánimo, así como de su género. Facilitando además la obtención de los resultados de dicho análisis de forma auditiva. Además, para facilitar la interacción del usuario con la herramienta se ha puesto a disposición un sistema comandos de voz los cuales se analizan y determinan la acción a realizar en el sistema incluyendo entre ellas la opción de guardar la identidad de una persona conocida para ser utilizada posteriormente.

Considero que la solución presentada es válida porque cumple los objetivos iniciales planteados para solucionar el problema.

5.2 Resumen de las contribuciones

Anteriormente se ha detallado las contribuciones específicas del proyecto. En la tabla siguiente se va a mostrar un cómo se asocian estas contribuciones basándose en los requisitos planteados inicialmente en el proyecto.

Requisito	Categoría	Descripción	Contribución
F01	N/A	El modelo de reconocimiento facial debe ser capaz de identificar caras de personas	Utilizando <i>OpenCV</i> y <i>Haar Cascades</i> se ha obtenido un modelo que es capaz de identificar en una imagen posibles caras devolviendo una lista de marcos para cada cara detectada
F02	N/A	El modelo de clasificación de emociones debe ser capaz de	Utilizando una CNN entrenada para clasificar la emoción de una cara de

		identificar emociones de las personas identificadas	entrada el sistema es capaz de identificar las emociones de las caras detectadas
F03	N/A	El modelo de clasificación de género debe ser capaz de identificar el género de las personas identificadas	Utilizando una CNN entrenada para clasificar el género de una cara de entrada el sistema es capaz de identificar el género de las caras detectadas
F04	N/A	La herramienta debe ser capaz de dada una imagen de entrada aplicar los diferentes modelos y obtener un resultado del análisis	La herramienta es capaz de capturar imágenes a través de una webcam y utilizando los modelos anteriormente mencionados efectúa un análisis y devuelve los resultados de forma visual y auditiva
F05	N/A	La herramienta debe ser capaz de informar de los resultados al usuario de forma auditiva	El sistema utilizando una librería que convierte texto a audio reproduce los resultados del análisis efectuado
F06	N/A	La herramienta debe ser capaz de recibir instrucciones en forma de comandos de voz del usuario	El sistema consta de un menú auditivo que se activa al pulsar la tecla espacio y que guía al usuario sobre las distintas opciones del sistema facilitando la interacción
F07	N/A	La herramienta debe permitir almacenar en el sistema la identidad de la persona identificada	El sistema tiene una opción para guardar la identidad de una persona detectada. Cuando esta opción es seleccionada la cara es guardada como archivo de imagen y el nombre es

			capturado, transformado a texto y después convertido a audio una vez más para ser utilizado posteriormente
F08	N/A	La herramienta debe ser capaz de reconocer si la identidad de la persona identificada pertenece a alguna de las almacenadas en el sistema y representar en la salida dicha identidad	Utilizando una librería que codifica una cara de entrada en una serie de identificadores el sistema es capaz de comparar caras entre sí y determinar si estas pertenecen a la misma persona o no. A su vez en base si la identidad coincide con alguna de las existentes en el sistema es capaz de recuperar el nombre en formato texto y de audio para ser representado o reproducido.
NF01	Usabilidad	La herramienta debe ser capaz de recibir instrucciones a través del teclado	El sistema consta de una serie de opciones que pueden ser activadas directamente a través del teclado
NF02	Usabilidad	La herramienta debe mostrar los resultados del análisis visualmente	El sistema modifica la imagen de entrada en tiempo real para mostrar los resultados del análisis constantemente
NF03	Rendimiento	El tiempo de análisis de la imagen de entrada debe ser razonable	El sistema es capaz de realizar el proceso de captura, análisis y muestra visual de los resultados en tiempo real
NF04	Accesibilidad	El sistema debe permitir la interacción en varios idiomas	La herramienta tiene dos idiomas implementados y plenamente funcionales en el

			sistema. Además, consta de un sistema que permite la adición de nuevos idiomas.
NF05	Portabilidad	El sistema debe ser capaz de ejecutarse en varios entornos y aceptar imágenes de diferentes fuentes de entrada	El sistema es capaz de ejecutarse en diferentes sistemas operativos siempre que estos soporten Python y las librerías utilizadas. La captura de imágenes por defecto se realiza a través de la webcam, pero el sistema se puede modificar fácilmente para realizar el análisis para recibir esta información desde otras fuentes.

Tabla 14: Requisitos y contribuciones

5.3 Líneas de trabajo futuro.

La herramienta desarrollada da solución a unos problemas específicos para las personas ciegas, pero sin duda la tecnología actual permite ir más allá. A continuación, se incluyen una serie de ideas de ampliación del sistema creado que permita mejorarlo.

Algunas de las ideas planteadas están actualmente en desarrollo por lo que puede que la versión final de la herramienta las incluya.

- **Permitir capturar como entrada imágenes capturadas del escritorio del ordenador.**

Por el momento las imágenes capturadas son a través de la webcam que permite enfocar los objetos a analizar, pero en el caso de que estos objetos sean imágenes existentes en el ordenador del usuario, ya sea en forma de archivo o a través de alguna aplicación instalada como puede ser Skype una webcam no es el mejor modo de capturar estas imágenes ya que implica tener que apuntar con ella a la pantalla.

- **Permitir analizar el contenido de las imágenes de entrada para catalogar escenas.**

Actualmente el sistema se centra únicamente en detectar caras y analizar sus emociones y género (opción WHO del sistema). Sería deseable que el sistema además fuera capaz de describir escenas o acciones en las imágenes sin necesidad de que estas fueran de un tipo concreto, es decir, desarrollar una opción WHAT que se pueda utilizar para describir de forma general una imagen.

- **Mejorar los modelos clasificatorios.**

La precisión del modelo de detección de emociones es de un 66% de éxito basado en los datos de prueba. Esto es claramente no óptimo por lo que se plantea mejorar la precisión de dicho modelo e incluso añadir más emociones a la lista de emociones a detectar.

- **Añadir nuevos modelos clasificatorios para las caras detectadas.**

Además de los ya existentes modelos de detección de género y emociones sería deseable utilizar más modelos que nos ayuden a definir con más precisión las características de las personas detectadas en la imagen. Un ejemplo de este tipo sería añadir al sistema un modelo de clasificación de edad.

- **Incrementar la portabilidad del sistema.**

Por el momento la herramienta está diseñada para ser utilizada en un ordenador. Éste puede ser un PC o un portátil siempre y cuando tenga Python y las librerías necesarias instaladas. Se pretende mejorar esto haciendo que el sistema sea capaz de funcionar en una Raspberry Pi (que no deja de ser un ordenador, pero mucho más pequeño y más portable). Otra opción planteada para mejorar la portabilidad del sistema es el desarrollo de aplicaciones de móviles que permitiría que la herramienta pueda ser utilizada mucho más fácilmente en cualquier sitio.

- **Aumentar la lista de idiomas soportados.**

Como se ha explicado en capítulos anteriores el sistema soporta múltiples idiomas así que la implementación de nuevos idiomas es una tarea relativamente sencilla.

Y estas son algunas de las ideas planteadas de cara a un trabajo futuro, aunque como se ha comentado anteriormente algunas están en desarrollo actualmente por lo que la versión final de la herramienta puede que incluya alguna de ellas.

CAPÍTULO 6

REFERENCIAS Y ENLACES

- Alex Krizhevsky, I. S. (2012). ImageNet Classification with Deep Convolutional. 9. Obtenido de <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- Anaconda. (2017). *Anaconda*. Recuperado el Abril de 2017, de Continuum: <https://www.continuum.io/>
- Anaconda. (2017). *Managing environments*. Recuperado el Abril de 2017, de <https://conda.io/>: <https://conda.io/docs/using/envs.html>
- Anthony Zhang, A. C. (2014). *SpeechRecognition*. Recuperado el Mayo de 2017, de <https://pypi.python.org/>: <https://pypi.python.org/pypi/SpeechRecognition/>
- Arriaga, O. (2017). *face_classification*. Obtenido de <https://github.com/>: https://github.com/oarriaga/face_classification
- Arriaga, O. (2017). *models.py*. Recuperado el Mayo de 2017, de https://github.com/oarriaga/face_classification/: https://github.com/oarriaga/face_classification/blob/master/src/models.py
- Convolution Matrix*. (2014). Obtenido de <https://docs.gimp.org/>: <https://docs.gimp.org/en/plugin-convmatrix.html>
- Diederik P. Kingma, J. L. (2017). *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*. Obtenido de <https://arxiv.org/>: <https://arxiv.org/pdf/1412.6980.pdf>
- Dlib C++ Library*. (2017). Obtenido de <http://dlib.net/>: <http://dlib.net/>
- Durette, P.-N. (2016). *Google Text to Speech*. Obtenido de <https://github.com/>: <https://github.com/pndurette/gTTS>
- Elisfm. (2009). *Elementos de un sistema de reconocimiento de patrones básico*. Recuperado el Mayo de 2017, de <https://commons.wikimedia.org/>: https://commons.wikimedia.org/wiki/File:Pattern_5.JPG
- face_recognition* 0.1.6. (2017). Obtenido de <https://pypi.python.org/>: https://pypi.python.org/pypi/face_recognition/0.1.6

- Geitgey, A. (13 de Junio de 2016). *Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks*. Recuperado el Abril de 2017, de <https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721>
- Google. (2017). *Cloud Speech API*. Recuperado el Junio de 2017, de Cloud Speech API: <https://cloud.google.com/speech/>
- Google. (2017). *Google Cloud Speech API Documentation*. Obtenido de <https://cloud.google.com/speech/docs/>
- Google. (2017). <https://trends.google.com>. Recuperado el Junio de 2017, de <https://trends.google.com/trends/explore?date=all&q=machine%20learning,pattern%20recognition,deep%20learning&hl=en-US>
- Google. (2017). *TensorFlow*. Recuperado el Abril de 2017, de <https://www.tensorflow.org/>
- Jupyter. (2017). <http://jupyter.org/>. Recuperado el Abril de 2017, de <http://jupyter.org/>
- Kaggle. (2013). *Challenges in Representation Learning: Facial Expression Recognition Challenge*. Recuperado el Mayo de 2017, de <https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data>
- Kaiming He, G. G. (2017). *Mask R-CNN*. Recuperado el Mayo de 2017, de <https://arxiv.org/pdf/1703.06870.pdf>
- Kairos. (2017). *Face Recognition: Kairos vs Microsoft vs Google vs Amazon vs OpenCV*. Obtenido de <https://www.kairos.com/blog/face-recognition-kairos-vs-microsoft-vs-google-vs-amazon-vs-opencv>
- Karn, U. (9 de Agosto de 2016). *A Quick Introduction to Neural Networks*. Recuperado el Abril de 2017, de <https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/>

- Karn, U. (11 de Agosto de 2016). *An Intuitive Explanation of Convolutional Neural Networks*. Recuperado el Abril de 2017, de <https://ujjwalkarn.me/https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>
- Karpathy, A. (2016). *CS231n Convolutional Neural Networks for Visual Recognition*. Obtenido de <http://cs231n.github.io/>: <http://cs231n.github.io/convolutional-networks/#pool>
- Karpathy, A. (2016). *CS231n Convolutional Neural Networks for Visual Recognition*. Recuperado el Abril de 2017, de <http://cs231n.github.io/>: <http://cs231n.github.io/neural-networks-1/>
- Karpathy, A. (2017). Recuperado el Abril de 2017, de <http://cs.stanford.edu/http://cs.stanford.edu/people/karpathy/neuraltalk2/demo.html>
- Keras. (2017). *Keras: The Python Deep Learning library*. Recuperado el Mayo de 2017, de <https://keras.io/>: <https://keras.io/>
- Kernel image processing*. (2017). Recuperado el Mayo de 2017, de [https://en.wikipedia.org/https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/https://en.wikipedia.org/wiki/Kernel_(image_processing))
- Malik, R. G. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation*. Recuperado el Abril de 2017, de <https://arxiv.org/https://arxiv.org/pdf/1311.2524.pdf>
- Nitish Srivastava, G. H. (2014). *Dropout: A Simple Way to Prevent Neural Networks from*. Recuperado el Abril de 2017, de <http://www.jmlr.org/http://www.jmlr.org/papers/volume15/srivastava14a.old/source/srivastava14a.pdf>
- Open CV. (2016). *facedetect.py*. Obtenido de <https://github.com/opencv/opencv/https://github.com/opencv/opencv/blob/master/samples/python/facedetect.py>
- Open CV. (2017). *Face Detection using Haar Cascades*. Recuperado el Mayo de 2017, de http://docs.opencv.org/http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html
- Open CV. (2017). *Haar Feature-based Cascade Classifier for Object Detection*. Recuperado el Mayo de 2017, de http://docs.opencv.org/http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html

- Open CV. (2017). *Open CV*. Recuperado el Mayo de 2017, de <http://opencv.org/>: <http://opencv.org/>
- Parthasarathy, D. (2017). *A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN*. Recuperado el Mayo de 2017, de <https://blog.athelas.com/>: <https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4>
- Paul Viola, M. J. (2001). *Rapid Object Detection using a Boosted Cascade of Simple*. Recuperado el Abril de 2017, de <http://cgит.nutn.edu.tw>: http://cgит.nutn.edu.tw:8080/cgit/PaperDL/uang_09032621564947.pdf
- Pete Shinnors, R. D. (2017). *Pygame*. Recuperado el Mayo de 2017, de <https://pypi.python.org/>: <https://pypi.python.org/pypi/Pygame>
- Peter Funk, J. H.-A. (2017). *gettext - Multilingual internationalization services*. Recuperado el Mayo de 2017, de <https://docs.python.org/>: <https://docs.python.org/3/library/gettext.html>
- Poedit. (2017). *Poedit - Gettext Translations Editor*. Recuperado el Mayo de 2017, de <https://poedit.net/>: <https://poedit.net/>
- Python. (2017). *Python*. Recuperado el Abril de 2017, de <https://www.python.org/>.
- Rasmus Rothe, R. T. (2015). *IMDB-WIKI – 500k+ face images with age and gender labels*. Recuperado el Mayo de 2017, de <https://data.vision.ee.ethz.ch/>: <https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/>.
- Raspberry Pi. (2017). *Raspberry Pi*. Recuperado el Mayo de 2017, de Raspberry Pi: <https://www.raspberrypi.org/>
- Ross Girshick, M. R. (2015). *Fast R-CNN*. Recuperado el Mayo de 2017, de <https://arxiv.org/>: <https://arxiv.org/pdf/1504.08083.pdf>
- Schapire, Y. F. (1997). *A Decision-Theoretic Generalization of On-Line Learning*. Recuperado el Mayo de 2017, de <http://www.face-rec.org/>: http://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf
- Shaoqing Ren, K. H. (2016). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. Obtenido de <https://arxiv.org/pdf/1506.01497v3.pdf>

- Simple DirectMedia Layer*. (2017). Recuperado el Junio de 2017, de SDL: <https://www.libsdl.org/>
- Sweigart, A. (2014). *Translate Your Python 3 Program with the gettext Module*. Obtenido de <https://inventwithpython.com/>: <https://inventwithpython.com/blog/2014/12/20/translate-your-python-3-program-with-the-gettext-module/>
- University of Massachusetts. (2017). *Labeled Faces in the Wild*. Obtenido de <http://vis-www.cs.umass.edu>: <http://vis-www.cs.umass.edu/lfw/>
- Wikipedia. (2017). *Artificial Neuron*. Recuperado el Junio de 2017, de <https://en.wikipedia.org/>: https://en.wikipedia.org/wiki/Artificial_neuron
- Yan LeCun, L. B. (1998). Gradient-Based Learning Applied to Document Recognition. Obtenido de <http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>

CAPÍTULO 7

ANEXOS

7.1 Instrucciones de instalación

A continuación, se explica el proceso para ejecutar la herramienta. Al estar desarrollada en Python no debería haber problema para ejecutarla en entornos Windows, macOS o Linux. Aunque no es estrictamente necesario las instrucciones presuponen que el entorno a utilizar tiene Anaconda (Anaconda, 2017) instalado. Otro requisito necesario es que si disponga de una web cam.

1. Descargar e instalar Anaconda:

```
https://www.continuum.io/downloads
```

2. Si se quiere hacer uso de la interacción a través de la voz del usuario es necesario realizar el proceso explicado en el apartado 7.2. Si se decide no hacerlo la herramienta sólo permitirá interacción a través del teclado.

3. Descargar o clonar el repositorio:

```
https://github.com/JosuVicente/facial_and_characteristics_recognition_with_speech_support
```

4. Crear un entorno conda a partir del archivo *environment/tfm.yml* del repositorio (Anaconda, 2017):

```
conda env create -f environment/tfm.yml
```

5. Activar el entorno tfm creado:

```
activate tfm
```

6. Posicionarse en la carpeta *src* y ejecutar *main.py*:

```
python main.py
```

7.2 Instrucciones para habilitar la interacción por voz

Para hacer uso de la interacción por voz es necesario disponer de una cuenta en Google Cloud Platform y habilitar el uso del Google Cloud Speech API. Google permite probarlo gratis durante 12 meses al conceder 300 dólares (<https://cloud.google.com/pricing/>). Cuando el

periodo de prueba expire el coste es de 0,0006 por cada 15 segundos de audio a partir de los primeros 60 minutos.

Las instrucciones para activar el servicio se encuentran detalladamente explicadas en la siguiente página: <https://cloud.google.com/speech/docs/getting-started>.

Una vez el servicio esté activado es necesario obtener una clave en un archivo JSON, las instrucciones se encuentran aquí: <https://cloud.google.com/speech/docs/auth>.

El archivo JSON generado deberá renombrarse a “GoogleCloudSpeechKey.json” y guardarse en la carpeta /files del proyecto:

```
files/GoogleCloudSpeechKey.json
```

Una vez hecho esto la herramienta es capaz de recibir comandos de voz.

7.3 Paquetes Python

A continuación, se muestra un detalle completo de los paquetes utilizados en este proyecto incluyendo las dependencias de estos:

```
name: tfm
channels:
- menpo
- conda-forge
- defaults
dependencies:
- bleach=1.5.0=py35_0
- bokeh=0.12.5=py35_2
- boto3=1.4.4=py35_0
- botocore=1.5.55=py35_0
- ca-certificates=2017.4.17=0
- certifi=2017.4.17=py35_0
- cffi=1.10.0=py35_0
- chardet=3.0.2=py35_1
- click=6.7=py35_0
- cloudpickle=0.3.1=py35_0
- colorama=0.3.9=py35_0
- cycler=0.10.0=py35_0
- dask=0.15.0=py35_0
- decorator=4.0.11=py35_0
- distributed=1.17.0=py35_0
- docutils=0.13.1=py35_0
```

```
- entrypoints=0.2.3=py35_1
- eventlet=0.21.0=py35_0
- flask=0.12=py35_0
- flask-socketio=2.8.5=py35_0
- freetype=2.7=vc14_1
- greenlet=0.4.12=py35_0
- h5py=2.7.0=np113py35_1
- hdf5=1.8.18=vc14_0
- heapdict=1.0.0=py35_0
- html5lib=0.999=py35_0
- icu=58.1=vc14_1
- idna=2.5=py35_0
- imageio=2.1.2=py35_0
- ipykernel=4.6.1=py35_0
- ipython=6.1.0=py35_0
- ipython_genutils=0.2.0=py35_0
- itsdangerous=0.24=py35_1
- jedi=0.10.0=py35_0
- jinja2=2.9.5=py35_0
- jmespath=0.9.2=py35_0
- jpeg=9b=vc14_0
- jsonschema=2.5.1=py35_0
- jupyter=1.0.0=py35_0
- jupyter_client=5.0.1=py35_0
- jupyter_console=5.1.0=py35_0
- jupyter_core=4.3.0=py35_0
- libpng=1.6.28=vc14_0
- libtiff=4.0.6=vc14_7
- locketch=0.2.0=py35_1
- markupsafe=1.0=py35_0
- matplotlib=2.0.2=np113py35_0
- mistune=0.7.4=py35_0
- msgpack-python=0.4.8=py35_0
- nbconvert=5.2.1=py35_1
- nbformat=4.3.0=py35_0
- networkx=1.11=py35_0
- notebook=5.0.0=py35_0
- olefile=0.44=py35_0
- openssl=1.0.2h=vc14_3
- packaging=16.8=py35_0
- pandas=0.20.2=np113py35_1
- pandoc=1.19.2=0
- pandocfilters=1.4.1=py35_0
- partd=0.3.8=py35_0
```

```
- patsy=0.4.1=py35_0
- pickleshare=0.7.3=py35_0
- pillow=4.1.1=py35_0
- pip=9.0.1=py35_0
- prompt_toolkit=1.0.14=py35_0
- psutil=5.2.1=py35_0
- pycparser=2.17=py35_0
- pygments=2.2.0=py35_0
- pyopenssl=16.2.0=py35_0
- pyparsing=2.2.0=py35_0
- pyqt=5.6.0=py35_4
- pysocks=1.6.7=py35_0
- python=3.5.2=5
- python-dateutil=2.6.0=py35_0
- python-engineio=1.5.2=py35_0
- python-socketio=1.7.4=py35_0
- pytz=2017.2=py35_0
- pywavelets=0.5.2=np113py35_0
- pyyaml=3.12=py35_1
- pyzmq=16.0.2=py35_2
- qt=5.6.2=vc14_1
- qtconsole=4.3.0=py35_0
- requests=2.17.3=py35_0
- s3fs=0.1.1=py35_0
- s3transfer=0.1.10=py35_1
- scikit-image=0.13.0=np113py35_0
- seaborn=0.7.1=py35_0
- setuptools=33.1.1=py35_0
- simplegeneric=0.8.1=py35_0
- sip=4.18=py35_1
- six=1.10.0=py35_1
- sortedcontainers=1.5.3=py35_0
- tblib=1.3.2=py35_0
- testpath=0.3=py35_0
- toolz=0.8.2=py35_0
- tornado=4.5.1=py35_0
- traitlets=4.3.2=py35_0
- urllib3=1.21.1=py35_1
- vc=14=0
- vs2015_runtime=14.0.25420=0
- wcwidth=0.1.7=py35_0
- webencodings=0.5=py35_0
- werkzeug=0.11.10=py35_0
- wheel=0.29.0=py35_0
```

```
- win_inet_pton=1.0.1=py35_1
- win_unicode_console=0.5=py35_0
- wincertstore=0.2=py35_0
- yaml=0.1.6=vc14_0
- zict=0.1.2=py35_0
- zlib=1.2.11=vc14_0
- asn1crypto=0.22.0=py35_0
- cryptography=1.8.1=py35_0
- mkl=2017.0.1=0
- numpy=1.13.0=py35_0
- scikit-learn=0.18.1=np113py35_1
- scipy=0.19.0=np113py35_0
- statsmodels=0.8.0=np113py35_0
- ipywidgets=5.1.5=py35_0
- opencv3=3.1.0=py35_0
- widgetsnbextension=1.2.3=py35_1
- pip:
  - alabaster==0.7.10
  - astroid==1.5.3
  - babel==2.4.0
  - cachetools==2.0.0
  - cx-freeze==5.0.2
  - dill==0.2.6
  - dlib==19.4.0
  - enum-compat==0.0.2
  - face-recognition==0.1.14
  - face-recognition-models==0.1.3
  - future==0.16.0
  - gapic-google-cloud-speech-v1==0.15.3
  - google-api-python-client==1.6.2
  - google-auth==1.0.1
  - google-auth-httpplib2==0.0.2
  - google-cloud-core==0.24.1
  - google-cloud-speech==0.25.1
  - google-gax==0.15.13
  - googleapis-common-protos==1.5.2
  - grpcio==1.3.5
  - gtts==1.2.0
  - gtts-token==1.1.1
  - httpplib2==0.10.3
  - imagesize==0.7.1
  - ipython-genutils==0.2.0
  - isort==4.2.15
  - jupyter-client==5.0.1
```

```
- jupyter-console==5.1.0
- jupyter-core==4.3.0
- keras==2.0.3
- lazy-object-proxy==1.3.1
- mccabe==0.6.1
- moviepy==0.2.3.2
- numpydoc==0.6.0
- oauth2client==4.1.1
- pep8==1.7.0
- ply==3.8
- prompt-toolkit==1.0.14
- proto-google-cloud-speech-v1==0.15.3
- protobuf==3.3.0
- pyasn1==0.2.3
- pyasn1-modules==0.0.9
- pyaudio==0.2.9
- pyflakes==1.5.0
- pygame==1.9.3
- pyinstaller==3.2.1
- pylint==1.7.1
- pypiwin32==219
- pyqt5==5.8.2
- pyreadline==2.1
- qtawesome==0.4.4
- qtpy==1.2.1
- rope-py3k==0.9.4.post1
- rsa==3.4.2
- snowballstemmer==1.2.1
- sortedcollections==0.5.3
- speechrecognition==3.6.5
- sphinx==1.6.2
- sphinxcontrib-websupport==1.0.1
- spyder==3.1.4
- statistics==1.0.3.5
- tensorflow==1.1.0
- theano==0.9.0
- tqdm==4.11.2
- unicode==0.4.20
- writemplate==3.0.0
- win-inet-pton==1.0.1
- win-unicode-console==0.5
- wrapt==1.10.10
```

7.4 Script de generación de archivos de audio utilizados

```
from gtts import gTTS
LANG_PATH = '../lang/{0}/speech/{1}.mp3'

tts = gTTS(text='Se ha detectado más de una persona, inténtelo de nuevo con una
persona sólo por favor', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'more_than_one_face'))
tts = gTTS(text='There appears to be more than one person, try again with one
person only please', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'more_than_one_face'))

tts = gTTS(text='ha sido guardado correctamente', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'saved'))
tts = gTTS(text='has been saved correctly', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'saved'))

tts = gTTS(text='diga el nombre de la persona detectada o cancelar después del
pitido por favor', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'who'))
tts = gTTS(text='say the name of the person detected or cancel after the beep
please', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'who'))

tts = gTTS(text='guardando', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'saving'))
tts = gTTS(text='saving', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'saving'))

tts = gTTS(text='un momento por favor', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'one_moment'))
tts = gTTS(text='one moment please', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'one_moment'))

tts = gTTS(text='lo siento, no he entendido bien', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'not_understand'))
tts = gTTS(text='sorry, I didn't catch that', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'not_understand'))

tts = gTTS(text='cancelado', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'canceled'))
tts = gTTS(text='canceled', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'canceled'))
```



```
tts = gTTS(text='seleccione una opción. O diga: Comandos. Para oír una lista de
comandos sonoros. O: Teclas. Para oír una lista de comandos de entrada.',
lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'choose'))
tts = gTTS(text='select an option. Or say Options to hear a list of available
commands. Or say Keys to hear a list of available keys', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'choose'))

tts = gTTS(text='seleccione una opción', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'choose_short'))
tts = gTTS(text='select an option', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'choose_short'))

tts = gTTS(text='Diga: ¿Quién? Para obtener una descripción de las personas en la
imagen. Diga: ¿Qué? Para obtener una descripción general de la imagen. Diga:
Guardar. Para guardar en el sistema el nombre de la persona en la imagen. Diga:
Idioma. Para cambiar el idioma al siguiente disponible. Diga: Cancelar. Para
continuar o diga: Repetir: Para repetir las opciones. ', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'commands'))
tts = gTTS(text='Say: Who. To get a description of the people in the image. Say:
What. To get a general description of the image. Say: Save. To save the name of the
person in the image. Say: Language. To change the language. Say: Cancel. To
continue. Say: Repeat. To repeat the list of available options', lang='en',
slow=False)
tts.save(LANG_PATH.format('en', 'commands'))

tts = gTTS(text='Pulse "A". Para obtener una descripción de las personas en la
imagen. Pulse "Z". Para obtener una descripción general de la imagen. Pulse "S".
Para guardar en el sistema el nombre de la persona en la imagen. Pulse "L". Para
cambiar el idioma al siguiente disponible. Pulse "Q". Para continuar o pulse: "R":
Para repetir las opciones. ', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'keys'))
tts = gTTS(text='Press "A". To get a description of the people in the image. Press
"Z". To get a general description of the image. Press "S". To save the name of the
person in the image. Press "L". To change the language. Press "Q". To continue.
Press "R". To repeat the list of available options', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'keys'))

tts = gTTS(text='Idioma cambiado a español', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'lang_change'))
tts = gTTS(text='Language has been changed to english', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'lang_change'))

tts = gTTS(text='De acuerdo', lang='es', slow=False)
```

```
tts.save(LANG_PATH.format('es', 'ok'))
tts = gTTS(text='OK', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'ok'))

tts = gTTS(text='Lo siento, no te he entendido', lang='es', slow=False)
tts.save(LANG_PATH.format('es', 'sorry_understand'))
tts = gTTS(text='Sorry, I did not get that', lang='en', slow=False)
tts.save(LANG_PATH.format('en', 'sorry_understand'))

tts = gTTS(text='¿Quieres que repita las opciones disponibles?', lang='es',
slow=False)
tts.save(LANG_PATH.format('es', 'repeat_options'))
tts = gTTS(text='Do you want me to repeat the available options?', lang='en',
slow=False)
tts.save(LANG_PATH.format('en', 'repeat_options'))

tts = gTTS(text='Lo siento, no soy capaz de describir la imagen', lang='es',
slow=False)
tts.save(LANG_PATH.format('es', 'no_image'))
tts = gTTS(text='Sorry, I cannot understand what's in the image', lang='en',
slow=False)
tts.save(LANG_PATH.format('en', 'no_image'))
```