**UNIVERSIDAD POLITÉCNICA DE MADRID**

**ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES**

**Aplicación de técnicas de visión artificial y Machine Learning en el desarrollo de algoritmos para el reconocimiento en vivo de expresiones faciales humanas e interpretación del lenguaje de señas de sordomudo americano**

**Autor**: Pedro José Zampella

**Tutor**: Sergio Domínguez

Madrid, Julio 2019

**Máster Universitario en Automática y Robótica**

## RESUMEN

El presente Trabajo Final de Máster expone el desarrollo de dos programas: uno capaz de clasificar expresiones faciales y otro capaz de clasificar señas del lenguaje de sordomudos (en versión americana), ambos utilizando técnicas de visión artificial y *Machine Learning*.

El alcance del proyecto incluye el diseño de los algoritmos capaces de capturar las imágenes obtenidas mediante la cámara integrada de un ordenador portátil, su tratamiento y adecuación para interpretación, así como el entrenamiento e implementación de las redes neuronales que realizan la clasificación de las imágenes, todo esto desarrollado en lenguaje Python.

Además, se creó una interfaz ejecutable tanto en sistema operativo Windows como Linux, que permite al usuario ejecutar cualquiera de los dos programas, indicar el número máximo de objetivos a clasificar en cada cuadro de vídeo, y que como resultado muestra en vivo la imagen captada por la cámara, donde se identifica y rastrea el objeto de interés, junto a la clasificación de este, según sea el caso.

El hecho de que se utilice el lenguaje Python durante su programación permitirá a estos programas ser utilizados como servicios en la nube que permitan a un robot humanoide utilizarlos libremente para demostraciones, o para permitir mayor eficacia durante su comunicación con seres humanos.

**Autor:** Pedro José Zampella

**Tutor empresarial:** Dr. Raúl Pérula-Martínez

**Tutor Académico:** Sergio Domínguez

**Palabras clave:** Inteligencia Artificial, *Machine Learning*, Perceptrón Multicapa, Red Neuronal Convolucional, Python, Visión Artificial, Seguimiento de Objetos.

## ÍNDICE DE CONTENIDO

[RESUMEN ii](#_Toc12801625)

[ÍNDICE DE CONTENIDO iii](#_Toc12801626)

[ÍNDICE DE FIGURAS v](#_Toc12801627)

[ABREVIATURAS Y ACRÓNIMOS vi](#_Toc12801628)

[1. INTRODUCCIÓN 7](#_Toc12801629)

[1.1. Contexto 7](#_Toc12801630)

[1.2. Estructura del documento 8](#_Toc12801631)

[2. MOTIVACIÓN Y OBJETIVO 10](#_Toc12801632)

[2.1. Motivación 10](#_Toc12801633)

[2.2. Objetivo general 11](#_Toc12801634)

[2.3. Objetivos específicos 11](#_Toc12801635)

[2.3.1. Reconocimiento de expresiones faciales humanas 11](#_Toc12801636)

[2.3.2. Interpretación del lenguaje de sordomudo versión americana 13](#_Toc12801637)

[3. ESTADO DEL ARTE 15](#_Toc12801638)

[3.1. Machine Learning y redes neuronales artificiales 15](#_Toc12801639)

[3.1.1. Perceptrón multicapa 17](#_Toc12801640)

[3.1.2. Mapa autoorganizado 18](#_Toc12801641)

[3.1.3. Red neuronal convolucional 19](#_Toc12801642)

[3.2. Visión artificial 22](#_Toc12801643)

[3.2.1. Seguimiento de objetos 22](#_Toc12801644)

[3.2.2. Clasificador de rostros 23](#_Toc12801645)

[3.2.3. Análisis de componente principal 24](#_Toc12801646)

[3.3. Lenguaje de programación Python 25](#_Toc12801647)

[3.3.1. Librerías Tensorflow y Keras 25](#_Toc12801648)

[3.3.2. OpenCV 26](#_Toc12801649)

[3.4. Proyectos similares 26](#_Toc12801650)

[3.4.1. Detecting Skin in Images & Video Using Python and OpenCV 26](#_Toc12801651)

[3.4.2. Clasificación de número escritos a mano con redes neuronales 27](#_Toc12801652)

[3.4.3. Sign Language Recognition with Unsupervised Feature Learning 28](#_Toc12801653)

[4. METODOLOGÍA Y DESARROLLO 31](#_Toc12801654)

[4.1. Elección del lenguaje de programación 31](#_Toc12801655)

[4.1.1. Librerías utilizadas 31](#_Toc12801656)

[4.2. Obtención de bases de datos para entrenamiento de redes neuronales 32](#_Toc12801657)

[4.2.1. Base de datos de rostros expresando emociones 33](#_Toc12801658)

[4.2.2. Base de datos de manos haciendo señas del lenguaje de sordomudos 34](#_Toc12801659)

[4.3. Preprocesamiento del dataset 35](#_Toc12801660)

[4.3.1. Preprocesamiento de imágenes de rostros 35](#_Toc12801661)

[4.3.2. Preprocesamiento de imágenes de señas 37](#_Toc12801662)

[4.4. Entrenamiento de redes neuronales artificiales 38](#_Toc12801663)

[4.4.1. Diseños previos de clasificadores 38](#_Toc12801664)

[4.4.2. Entrenamiento del clasificador de expresiones faciales 40](#_Toc12801665)

[4.4.3. Entrenamiento del clasificador señas 45](#_Toc12801666)

[4.5. Clasificación en vivo 47](#_Toc12801667)

[5. RESULTADOS 54](#_Toc12801668)

[5.1. Pruebas post-entrenamiento 54](#_Toc12801669)

[5.2. Pruebas en vivo 55](#_Toc12801670)

[5.2.1. Clasificación de expresiones faciales humanas 55](#_Toc12801671)

[5.2.2. Clasificación de señas del lenguaje de sordomudo americano 56](#_Toc12801672)

[6. POSIBLES APLICACIONES 59](#_Toc12801673)

[7. CONCLUSIONES Y REFLEXIONES FINALES 60](#_Toc12801674)

[8. MEJORAS Y FUTUROS DESARROLLOS 63](#_Toc12801675)

[9. BIBLIOGRAFÍA 64](#_Toc12801676)

[10. ANEXOS 66](#_Toc12801677)

## ÍNDICE DE FIGURAS

[Figura 1: Robots Pepper y Nao presentes en el wavespace de EY 10](#_Toc12801558)

[Figura 2: Ejemplo de sujetos mostrando diferentes expresiones faciales 12](#_Toc12801559)

[Figura 3: Alfabeto del lenguaje de señas americano 13](#_Toc12801560)

[Figura 4: Ejemplo de sobreentrenamiento y bajo entrenamiento 16](#_Toc12801561)

[Figura 5: Arquitectura del perceptrón multicapa 17](#_Toc12801562)

[Figura 6: Mapa autoorganizado durante proceso de entrenamiento 18](#_Toc12801563)

[Figura 7: Arquitectura de una red neuronal convolucional 20](#_Toc12801564)

[Figura 8: Funcionamiento de una neurona convolucional 21](#_Toc12801565)

[Figura 9: Uso de ventanas deslizantes para clasificación de rostros 23](#_Toc12801566)

[Figura 10: Ejemplo del uso de PCA para clasificación de muestras 24](#_Toc12801567)

[Figura 11: Detección de piel y filtrado en imagen 27](#_Toc12801568)

[Figura 12: Ejemplo de dígitos de la base de datos MNIST 27](#_Toc12801569)

[Figura 13: Métodos de procesamiento de imagen 29](#_Toc12801570)

[Figura 14: Muestras del dataset utilizado para identificación de emociones 33](#_Toc12801571)

[Figura 15: Alfabeto contenido en el dataset 35](#_Toc12801572)

[Figura 16: Diagrama de flujo de procesamiento de imágenes de rostros 36](#_Toc12801573)

[Figura 17: Diagrama de flujo del entrenamiento del clasificador de expresiones faciales 40](#_Toc12801574)

[Figura 18: Gráfica de la función RELU clásica y suavizada 43](#_Toc12801575)

[Figura 19: Gráfica de la función sigmoide 44](#_Toc12801576)

[Figura 20: Diagrama de flujo del entrenamiento del clasificador de señas 45](#_Toc12801577)

[Figura 21: Diagrama de flujo del clasificador en vivo 48](#_Toc12801578)

[Figura 22: Diagrama de flujo del procesamiento en vivo de imágenes 49](#_Toc12801579)

[Figura 23: Secuencia de imágenes mostrando procesamiento al identificar piel humana 51](#_Toc12801580)

[Figura 24: Tablas de resultados de pruebas después del entrenamiento 54](#_Toc12801581)

[Figura 25: Sujeto uno mostrando expresiones de “ira” y “disgusto” 55](#_Toc12801582)

[Figura 26: Sujeto dos mostrando expresiones de “tristeza” y “sorpresa” 55](#_Toc12801583)

[Figura 27: Sujeto tres mostrando expresiones de “alegría” y “miedo” 56](#_Toc12801584)

[Figura 28: Los tres sujetos de prueba mostrando la expresión “neutral” 56](#_Toc12801585)

[Figura 29: Sujeto mostrando las señas para las letras “A”, “F”, “I”, “L”, “O”, “P”, “Q”, “Y” 57](#_Toc12801586)

## ABREVIATURAS Y ACRÓNIMOS

CNN: Red neuronal convolucional (del inglés *Convolutional Neural Network*)

KNN: K vecinos más próximos (del inglés *K-Nearest Neighbor*)

MIL: Aprendizaje de múltiple instancia (del inglés *Multiple Instance Learning*)

ML: *Machine Learning* (Aprendizaje automático, en español)

MLP: Perceptrón multicapa (del inglés *Multilayer Perceptron*)

MNIST: (Base de datos) modificada del Instituto Nacional de Estándares y Tecnología (del inglés *Modified National Institute of Standards and Technology Database*)

PCA: Análisis de componentes principales (del inglés *Principal Component Analysis*)

SOM: Mapa autoorganizado (del inglés *Self-Organized Map*)

# INTRODUCCIÓN

### Contexto

Durante las últimas dos décadas, ha habido un crecimiento exponencial de la robótica en el ámbito industrial, comercial y social. Es cada vez más común encontrar robots haciendo trabajos repetitivos que antes eran realizados por seres humanos, quienes ahora se encuentran con el tiempo necesario para encargarse de tareas más creativas y que agregan un mayor valor a la sociedad.

Estos robots, a su vez, se van haciendo cada vez mejores en la realización de sus tareas, y hay industrias que en la actualidad ya no son concebibles sin robots: automotriz, aeroespacial, distribución, etc. Sin embargo, la introducción de los robots en la parte comercial y social ha sido un poco más lenta, principalmente debido a dos razones: primero, las habilidades necesarias en un robot en este ámbito son más complejas y requieren de un tacto que hasta hace pocos años se consideraba como únicamente humano; y segundo, los seres humanos sentimos miedo a ser sustituidos.

En la actualidad, se pretenden desarrollar herramientas que permitan a los robots debilitar ambas barreras a través de la comunicación directa con los seres humanos. En un principio, la comunicación entre un ser humano y un robot se hacía totalmente a través del código con el que se le programaba, de forma tal que sólo los técnicos podían enviar órdenes e interpretar respuestas. Este método de comunicación es suficiente en el caso de robots industriales, puesto que su interacción con el resto del personal en el área de trabajo debe ser mínimo o, en el mejor de los casos, no existente. Sin embargo, en el caso de los robots enfocados en el ámbito comercial y social la situación cambia radicalmente, ya que su principal función es la de interactuar con seres humanos, por lo que se espera que la comunicación sea a través de un lenguaje más natural.

Ya existen muchos trabajos que tratan la interpretación del lenguaje natural para robots, ya sea escrito o hablado. En 2014 se creó el primer programa no solo capaz de interpretar el lenguaje natural escrito, sino que fue capaz de pasar el test de Turing [1] (aunque algunos expertos no lo consideran cierto). Por otra parte, en la actualidad nuestra sociedad se ha inundado de programas capaces de interpretar nuestro lenguaje y ofrecernos respuestas, desde las contestadoras automáticas de las grandes compañías hasta los chatbots. Es por ello por lo que este Trabajo Final de Máster trata sobre el desarrollo de dos herramientas que permitirían la interacción de los robots de una forma diferente.

Primero, una herramienta capaz de ayudar al robot a interpretar las expresiones faciales del interlocutor, permitiendo así reducir la brecha entre las meras palabras expresadas por este, y un significado que puede variar dependiendo la intención que va por detrás de ellas.

Segundo, otra herramienta capaz de interpretar el lenguaje de señas de sordomudos en su versión americana, de forma tal que un robot pueda ser capaz de interpretar lo que una persona con déficit de audición intente comunicarle, sin la utilización de un teclado o pantalla táctil. Esta herramienta está más enfocada a robots humanoides, puesto que estos tienen una interacción directa con el público, en el que la inclusión de personas con dificultades en el habla puede marcar una diferencia positiva para la percepción de los robots sociales.

A lo largo de este trabajo, se explicarán las tecnologías y técnicas utilizadas durante su realización, así como sus posibles aplicaciones en casos de la vida real donde se le pueda sacar el máximo provecho posible.

### Estructura del documento

Este documento consta de una estructura donde se analiza y explica el desarrollo de este trabajo, comenzando por un repaso sobre las tecnologías utilizadas, pasando por la metodología seguida y cerrando con los resultados obtenidos y conclusiones finales.

Luego de la introducción que se está realizando en el primer capítulo, el segundo explicará la motivación para su realización con mayor detalle, así como una exposición clara de los objetivos que se buscan cumplir durante su desarrollo.

El tercer capítulo realizará una revisión del estado del arte de la tecnología actual sobre la que se está apoyando el presente desarrollo. Aquí también se mostrarán trabajos previos que en parte han inspirado algunas de las estrategias seguidas.

El capítulo cuatro busca mostrar los detalles de la metodología seguida durante el desarrollo del proyecto, la selección de técnicas y métodos utilizados durante el mismo, así como su integración para obtener cada uno de los programas finales.

El capítulo cinco expone los resultados obtenidos en la actual implementación, tanto durante el desarrollo de los programas como ejemplos de su aplicación real.

El sexto capítulo, séptimo y octavo capítulos incluyen las posibles aplicaciones y mejoras que se le pueden dar al proyecto, y en conjunto dan una idea sobre el futuro que pueda tener este proyecto, además de las conclusiones y reflexiones finales.

Finalmente, todos los documentos y las referencias utilizadas a lo largo de este trabajo serán indicadas al final, junto con el repositorio donde se puede encontrar el código de los programas en sí.

# MOTIVACIÓN Y OBJETIVO

Las principales motivaciones para la elección de este proyecto se explicarán a continuación, y de igual manera, tanto los objetivos generales como los objetivos específicos de este trabajo serán desglosados según su índole.

### Motivación

La consultora multinacional Ernst & Young, en su sede en Madrid, cuenta desde hace varios años con un Centro de Excelencia especializado en Inteligencia Artificial, conocido como *wavespace*, que es único en la empresa a nivel mundial, y en donde se trabajan principalmente proyectos de innovación especializados en automatización de procesos, desarrollo web, visión artificial, Big Data, visualización de datos, realidad virtual, realidad aumentada y robótica de humanoides, para clientes internos y externos, nacionales, europeos e internacionales.

Este centro tiene a su disposición dos robots en su auditorio de exposiciones, modelos Pepper y Nao, los cuales han sido modificados y reconfigurados con avanzados programas de visión artificial, reconocimiento de imágenes, identificación de personas, interpretación de lenguaje natural, identificación de emociones mediante el contenido del mensaje, traducción en vivo entre varios idiomas y mapeo para desplazamiento dentro del propio auditorio.



Figura 1: Robots Pepper y Nao presentes en el wavespace de EY

Estos robots son el centro de atención de clientes y demás empleados de la organización, por lo que se quiere seguir aumentando el abanico de posibilidades que puedan ofrecer al público.

Es por ello por lo que la propuesta de este proyecto, consistente en equiparlos con servicios capaces de mejorar la comunicación visual con seres humanos ha interesado a la junta directiva, razón por la cual ha puesto a disposición los recursos que permitan su desarrollo en cooperación con la Universidad Politécnica de Madrid.

### Objetivo general

El objetivo principal de este proyecto es el desarrollo de programas en Python basados en inteligencia artificial, *Machine Learning* y visión artificial, los cuales puedan ser utilizados por robots para mejorar la comunicación visual en vivo con seres humanos, y que puedan funcionar como servicios en la nube, de forma tal que sean extensibles a diferentes locaciones geográficas y en simultáneo.

### Objetivos específicos

De manera más específica, este proyecto busca enfocarse en la solución de dos problemas bien diferenciados, pero que comparten muchas características en común desde el punto de vista técnico, como lo son el reconocimiento de las expresiones faciales de seres humanos y la interpretación del lenguaje de sordomudos, en su versión americana, utilizando técnicas avanzadas de visión artificial y *Machine Learning*, o más específicamente, de redes neuronales artificiales.

A pesar de sus puntos en común, cada uno de estos programas será un módulo independiente al que se podrá acceder y ejecutar independientemente, siempre y cuando el hardware utilizada cumpla con unos requerimientos mínimos (principalmente, un dispositivo de vídeo de entrada).

A continuación, se especifican los objetivos de cada uno de los dos programas:

#### Reconocimiento de expresiones faciales humanas

Para el programa encargado del reconocimiento de las expresiones faciales humanas, se espera alcanzar los siguientes objetivos.

* Filtrar dentro de cada cuadro de vídeo los objetos cuyo color se encuentre dentro de los rangos de color de la piel humana.
* Identificar de entre los objetos filtrados como “piel humana”, aquellos que sean rostros humanos.
* Realizar el seguimiento cuadro a cuadro de los objetos identificados por la cámara, mientras se encuentren dentro del campo de visión de la misma.
* Entrenar una red neuronal artificial capaz de clasificar los rostros humanos, de acuerdo con las siguientes siete expresiones faciales:
  + Neutral
  + Alegría
  + Tristeza
  + Ira
  + Miedo
  + Desagrado
  + Sorpresa
* Clasificar los rostros observados (limitando su número para no disminuir desempeño de la ejecución) en el campo de visión de la cámara utilizando la red neuronal artificial mencionada anteriormente.
* Mostrar en vivo y sobre el vídeo la región de interés identificada y la clasificación realizada.

A pesar de que para este proyecto se desea mostrar el resultado en la pantalla, para su aplicación sólo se requeriría transmitir al robot el resultado de la clasificación, la cual sería utilizada como información de entrada para otros procesos más complejos.



Figura 2: Ejemplo de sujetos mostrando diferentes expresiones faciales

La figura anterior muestra cuatro sujetos demostrando diferentes expresiones faciales de forma actuada, con un fondo monocromático y sin objetos, tal y como se presentarán en el dataset de entrenamiento de la red neuronal artificial.

Se espera obtener un porcentaje de aciertos de al menos el 75% de los casos.

#### Interpretación del lenguaje de sordomudo versión americana

Se esperan alcanzar los siguientes objetivos para la interpretación del lenguaje de sordomudos, en su versión americana:

* Filtrar dentro de cada cuadro de vídeo los objetos cuyo color se encuentre dentro de los rangos de la piel humana.
* Identificar de entre los objetos filtrados como “piel humana”, aquellos que no correspondan con rostros humanos, los cuales se asumirán como manos.
* Realizar el seguimiento de los objetos identificados mientras se encuentren dentro del campo de visión de la cámara.
* Entrenar una red neuronal artificial capaz de clasificar las posiciones de las manos correspondientes a cada una de las letras del lenguaje de sordomudos (versión americana), con la excepción de la “J” y la “Z”, por no ser estáticas.
* Clasificar las señas realizadas por las manos observadas en el campo de visión de la cámara utilizando la red neuronal descrita anteriormente.
* Mostrar en vivo y sobre el vídeo la región de interés identificada y la clasificación realizada.

En este caso también se desea mostrar el resultado en la pantalla, a pesar de que para su aplicación sólo se requeriría transmitir al robot el resultado de la clasificación para ser utilizado en su interpretación.



Figura 3: Alfabeto del lenguaje de señas americano

La figura anterior muestra el lenguaje de señas utilizado para este proyecto, excluyendo aquellas que implican movimientos de la mano (“J” y “Z”) al requerir una metodología diferente a la que se está estudiando, y que por lo tanto escapan del alcance de este trabajo.

De igual manera, en este caso también se está utilizando un fondo monocromático y sin objetos, lo cual deberá ser replicado durante su uso en vivo.

Se espera obtener un porcentaje de aciertos de al menos el 80% de los casos.

# ESTADO DEL ARTE

Como se ha mencionado anteriormente, el presente proyecto está basado en dos programas diferenciables. Sin embargo, ambos programas tienen en común una estructura con dos módulos diferenciables: el desarrollo de modelos de aprendizaje automático o *Machine Learning*, y el procesamiento de imágenes usando técnicas de visión artificial. Ambos módulos fueron desarrollados en lenguaje Python.

### Machine Learning y redes neuronales artificiales

El *Machine Learning* (en español, aprendizaje automático) es una rama de las ciencias de la computación y de la inteligencia artificial, que consiste en desarrollar técnicas que permitan a programas informáticos aprender. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos [2].

Una de las técnicas de *Machine Learning* más utilizadas, y en la que se basó este proyecto, son las redes neuronales artificiales. Estas se definen como un modelo computacional inspirado en el comportamiento observado en su homólogo biológico. Consisten en un conjunto de unidades, denominadas neuronas artificiales, las cuales se encuentran conectadas entre sí, transmitiendo señales basado en ciertas condiciones. La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida [3].

En forma general, estos modelos se utilizan principalmente con dos objetivos: predicción y clasificación. Para ambos casos, la metodología utilizada es similar: se debe construir un dataset con una gran variedad de casos que se deben procesar. Este dataset se divide en una parte que será utilizada para el entrenamiento del algoritmo, y otra que será utilizada para validar los resultados de este.

Hay dos tipos de redes neuronales: supervisadas y no supervisadas. La principal diferencia entre ambas es que, en el caso de las redes neuronales supervisadas, el dataset incluye el valor de clasificación esperado para cada muestra, mientras que, en el caso no supervisado, dicho valor no se conoce.

Durante la etapa de entrenamiento, la conexión entre cada una de las neuronas artificiales varía su peso (el umbral a partir del cual dejará pasar la señal o no, es decir, se “activará”) dependiendo de si la salida se acercó al valor esperado o no, y este proceso se repetirá iterativamente hasta que el programador considere que el aprendizaje ha culminado. Luego, el modelo se evalúa utilizando la parte del dataset que no se usó para el entrenamiento, de forma tal que sea posible evaluar la precisión del este.

Es importante tomar en consideración que la decisión sobre si la etapa de entrenamiento ha culminado o no queda enteramente a discreción del programador, y la efectividad de la red neuronal puede depender en gran parte de su experticia: si se detiene el aprendizaje de la red neuronal antes de tiempo, sus resultados no serán confiables, ya que no logró analizar suficientes casos; por otro lado, si se alarga la etapa de aprendizaje demasiado tiempo, la red neuronal podría caer en el estado de “sobreentrenamiento”, y no ser confiable tampoco.

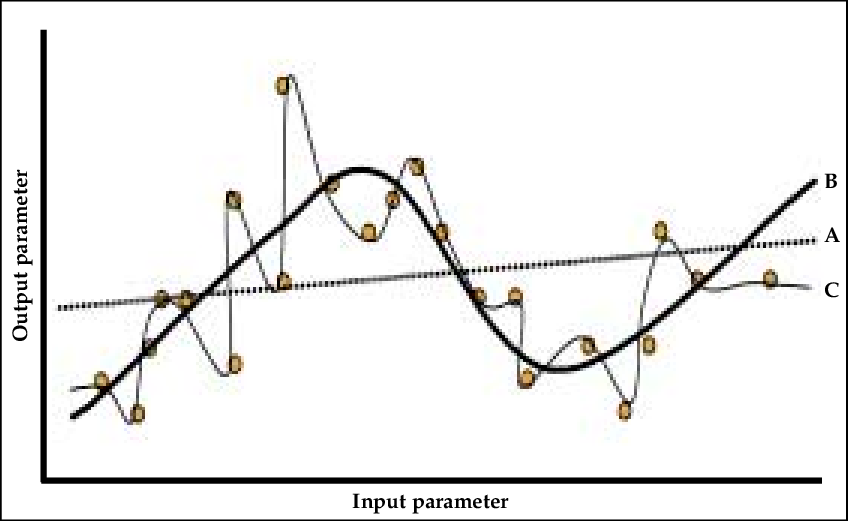


Figura 4: Ejemplo de sobreentrenamiento y bajo entrenamiento

En la figura anterior se pueden observar los tres posibles casos que se mencionaron anteriormente.

Para una función coseno con un ruido introducido, se tomaron 23 muestras, y se entrenaron tres redes neuronales para que encontraran la función original sin ruido:

* Caso A: esta red neuronal no fue entrenada por suficiente tiempo, por lo que, si bien muestra una línea de tendencia similar a la esperada, no es capaz de seguir la función original.
* Caso B: esta red neuronal fue entrenada por un tiempo suficiente, y se puede observar cómo sigue el valor de la función a pesar de casi no tocar ninguna muestra. Es decir, está logrando el objetivo de obtener la función original sin ruido.
* Caso C: esta red neuronal ha sido sobreentrenada, razón por la cual se ha visto afectada considerablemente por el ruido y se ha alejado de la forma de la función original. Se podría decir que en lugar de “aprender” de la muestra, lo que ha hecho ha sido “memorizarla”.

#### Perceptrón multicapa

Existen diversas morfologías para las redes neuronales, siendo el perceptrón multicapa (MLP) el más extendido en la actualidad.

Este consiste en una red neuronal artificial conformada por múltiples capas, lo cual le otorga la capacidad de resolver problemas que no son linealmente separables. Cada una de estas capas está conectada con la capa anterior a través de sus neuronas. Al igual que en el caso de su homóloga biológica, para que el estímulo pase de una neurona a otra, este debe alcanzar el umbral de activación. Este umbral de activación puede variar dependiendo del programador, pero las dos más utilizadas son las siguientes funciones sigmoides:

En el caso en el que las capas están totalmente interconectadas, cada salida de una neurona de la capa “i” es entrada de todas las neuronas de la capa “i+1”, aunque dependiendo del caso, este diseño puede variar ligeramente [4].

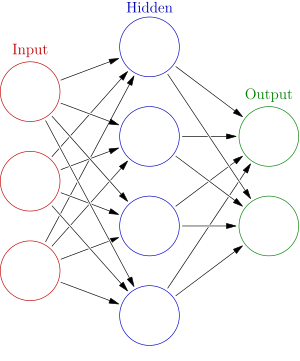


Figura 5: Arquitectura del perceptrón multicapa

Entre los parámetros que se suelen ajustar durante el entrenamiento, se encuentran:

* La función de activación (usualmente sigmoide): Se encarga de devolver una salida a partir de un valor de entrada cuando sobrepasa cierto umbral.
* El número de capas ocultas: Es el número de capas de neuronas dentro de la red que se encuentra entre la capa de entrada y la capa de salida.
* El número de neuronas en cada capa oculta: Es la cantidad de neuronas en cada una de estas capas, la cual puede o no variar de cada y capa.
* El tamaño del lote (*batch*, en inglés) que se utiliza en cada iteración: Es tamaño de cada subconjunto del dataset de entrada. Se utiliza cuando existe un gran número de elementos a ser utilizados durante el entrenamiento, lo cual puede producir problemas de rendimiento en el equipo utilizado.
* El número de ciclos: Es el número total de iteraciones que debe ocurrir hasta que todos los lotes que conforman el dataset hayan sido procesados.

Adicionalmente, el número de neuronas en la capa de entrada es definido por el número de dimensiones de cada una de las muestras, y el número de neuronas en la capa de salida, depende del número de dimensiones de la respuesta esperada.

#### Mapa autoorganizado

Los mapas autoorganizados (SOM, por sus siglas en inglés) son un tipo de red neuronal estrictamente no supervisada, que utiliza una función de vecindad para preservar las características topológicas del espacio de entrada, permitiendo visualizar vistas de baja resolución en datos de alta dimensión [5].

Asociado a cada neurona hay un vector de pesos, de la misma dimensión que los vectores de entrada, y con una posición en el mapa. Usualmente, las neuronas se disponen en una red de dos dimensiones, ya sea hexagonal o rectangular. Los mapas autoorganizados describen el mapeo de un espacio de mayor dimensión a uno de menor dimensión.

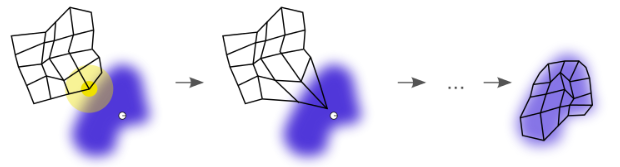


Figura 6: Mapa autoorganizado durante proceso de entrenamiento

El procedimiento para ubicar un vector del espacio de los datos en el mapa es encontrar la neurona con el vector de pesos más cercano (menor distancia métrica) al vector del espacio de los datos. Esto puede traer como consecuencia que dos puntos que son vecinos en el espacio de mayor dimensión puedan encontrarse lejos en el espacio dimensional inferior.

El entrenamiento utiliza el denominado aprendizaje competitivo. Cuando una matriz de datos es procesada, la distancia euclidiana a todos los vectores de pesos de las neuronas es calculada. La neurona cuyo vector de pesos sea más similar a la entrada es llamada unidad de mejor correspondencia, y tanto sus pesos como los de las neuronas cercanas a ella en la cuadrícula del SOM son ajustados hacia el vector de entrada. La magnitud de los cambios se decrece con el tiempo (en iteraciones) y con la distancia desde la unidad de mejor correspondencia.

Este proceso es repetido para cada vector de entrada durante un número de ciclos usualmente grande. La red va asociando las neuronas de salida con grupos o patrones en el conjunto de entrenamiento.

#### Red neuronal convolucional

Otra morfología muy utilizada en la actualidad es la red neuronal convolucional (CNN, en inglés). Este es un tipo de red neuronal donde las neuronas se corresponden a los campos receptivos imitando a sus equivalentes biológicos en la corteza visual del cerebro [6].

La función de activación es típicamente una capa de rectificación (RELU, en inglés), la cual no más que una función de activación simple y centrada en cero:

La morfología de esta red es una variación del perceptrón multicapa explicado anteriormente, y al ser aplicado a matrices bidimensionales, se muestran particularmente útiles en problemas de visión artificial, especialmente pero no limitado a casos de clasificación y segmentación de imágenes.

Su arquitectura consiste en múltiples capas de filtros convolucionales que pueden tener una o más dimensiones. Entre cada una de las capas, se puede añadir una función para realizar un mapeo causal no-lineal. Las primeras capas están compuestas de neuronas convolucionales y de reducción de muestreo, con el fin de extraer las características de la imagen en general, y no de cada uno de sus puntos. En las últimas capas se encuentran neuronas de perceptrón sencillas para realizar la clasificación final sobre las características extraídas.

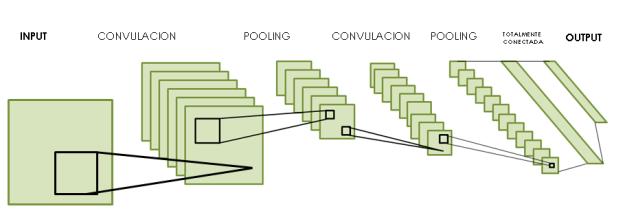


Figura 7: Arquitectura de una red neuronal convolucional

La fase de extracción de características es equivalente al proceso estimulante en las células de la corteza visual, y está compuesta de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. En la medida que se avanza a través de las capas, se reduce el número de neuronas, siendo las neuronas en capas finales menos sensibles a perturbaciones en los datos de entrada, y activándose cada vez por características más complejas [7].

Las neuronas convolucionales se diferencian de los perceptrones ya que son procesadores de matriz cuadrada que realizan una operación de convolución sobre un campo en 2D, en lugar de trabajar sobre un solo píxel. La salida de cada neurona se calcula de la siguiente manera:

Donde la salida **Yj** de una neurona **j** es una matriz que se calcula por medio de la combinación lineal de las neuronas **Yi** en la capa anterior **i**, cada una de ellas, operadas con el núcleo convolucional **K** correspondiente a esa conexión. Esta cantidad es sumada a una influencia **b** y luego se pasa por una función de activación  **(·)** no lineal.

Las neuronas de reducción de muestreo permiten aumentar la tolerancia de la red neuronal ante pequeños cambios entre dos imágenes (por ejemplo, una misma imagen, pero trasladada unos pocos píxeles). Para ello, estas neuronas utilizan operaciones de *subsampling*, principalmente *max-pooling*, que permiten obtener tomar un valor resumen de las características de una cierta área, reduciendo de esta manera el tamaño de los datos.

* *Subsampling*: es una operación constante que permite la reducción de la muestra evitando perder sus características principales. En CNN, usualmente la operación utilizada es el *max-pooling*.
* *Max-Pooling*: se utiliza para reducir el tamaño de la muestra, tomando una matriz cuadrada de un tamaño definido (2x2, 3x3…) a lo largo de toda la imagen, y sustituyendo el subconjunto de píxeles dentro de cada matriz por un único nuevo píxel con el valor máximo encontrado.

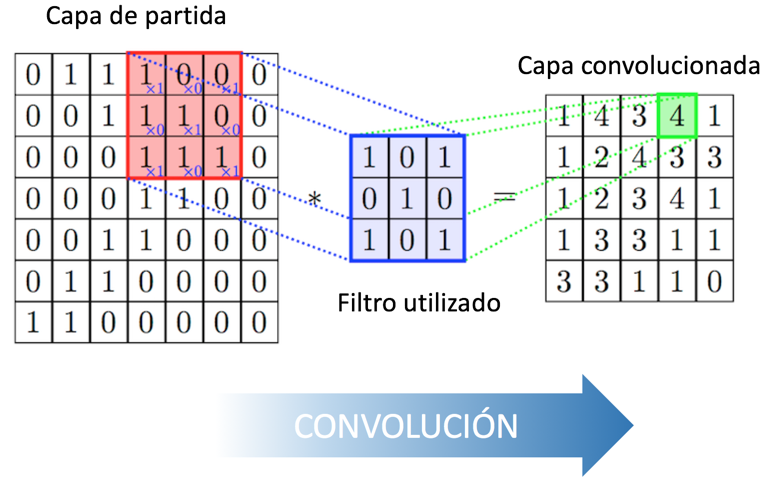


Figura 8: Funcionamiento de una neurona convolucional

Las neuronas de clasificación son la última etapa de la red neuronal convolucional, y su morfología es la de un MLP clásico de una sola capa, puesto que los datos que recibe ya han sido depurados y la clasificación se puede realizar de forma más genérica.

Entre los parámetros que se suelen ajustar durante el entrenamiento, se encuentran:

* La función de activación (usualmente una capa RELU, que puede ser suavizada o no).
* El número de capas ocultas.
* El número de neuronas en cada capa oculta.
* El tamaño del lote que se utiliza en cada iteración.
* El número de ciclos.
* El tamaño del filtro de convolución.
* El tamaño de la matriz de max-pooling.

De igual manera, el número de neuronas en la capa de entrada y de la capa de salida depende respectivamente del número de dimensiones de la imagen de entrada y del número posible de respuestas esperadas (o clases).

### Visión artificial

La visión artificial es una rama de las ciencias del computador que incluye métodos y técnicas para adquirir, procesar, analizar y comprender imágenes en el mundo real, y transformarlos en información numérica que pueda ser interpretada por un computador. Esta interpretación se logra a través de distintos campos como la geometría y la estadística, principalmente. La adquisición de los datos se consigue por varios métodos, que van desde archivos de imágenes simples, secuencias de vídeo grabado o en vivo, o hasta datos multidimensionales obtenidos por un escáner médico.

Entre las principales tecnologías que utilizan visión artificial, se encuentran:

* Limpieza de imágenes.
* Reconocimiento de objetos.
* Seguimiento de objetos (*tracking*).
* Reconstrucción de escenas.
* Segmentación de imágenes.

Durante el desarrollo de un algoritmo de visión artificial, se debe tomar en cuenta que, si bien la calidad y resolución de imagen ayuda a obtener mejores resultados, el procesamiento de estas imágenes va a consumir un mayor tiempo de procesamiento, por lo que se debe buscar un punto intermedio sobre el cual trabajar.

Una manera de mitigar este problema es realizar transformaciones sobre las imágenes para reducir la carga sobre el procesador. Algunas de estas transformaciones son:

* Reducción de la resolución de la imagen (*resize*).
* Recorte del área de interés (*crop*).
* Transformación de la imagen a escala de grises.
* Análisis de componente principal (PCA, por sus siglas en inglés).

#### Seguimiento de objetos

Otro punto importante es la utilización del seguimiento de objetos entre cuadros de vídeo sucesivos. Existen varias metodologías que se pueden aplicar para lograr este objetivo, y durante el desarrollo de este algoritmo se utilizó el *tracker* de tipo MIL (*Multiple Instance Learning*, en inglés).

Este tipo de *tracker* se entrena en tiempo real con grupos de ejemplos positivos y negativos del objeto, acumulados a lo largo de la ejecución. El cuadro delimitador suministrado inicialmente se toma como un ejemplo positivo para el objeto, y varios parches de imagen fuera del cuadro delimitador se tratan como fondo. Dado un nuevo cuadro de vídeo, el clasificador se ejecuta en cada píxel en la vecindad de la ubicación anterior y se registra la puntuación del tracker. La nueva ubicación del objeto es aquella en la que la puntuación es máxima para alguno de los ejemplos positivos [8].

#### Clasificador de rostros

En la actualidad existen muchos algoritmos de detección de rostros. Para este caso, se eligió el LPE Haar Cascade al estar disponible en la librería OpenCV y tener un alto porcentaje de aciertos.

El LPE Haar Cascade es un algoritmo de detección de objetos por *Machine Learning* que se utiliza para identificar rostros frontales en una imagen o vídeo, y se basa en el concepto de características propuesto por Paul Viola y Michael Jones en 2001.

Este clasificador en cascada consiste en una colección de etapas, donde cada etapa consiste en un conjunto de “aprendices” débiles. Los aprendices débiles son simples clasificadores denominados nodos de decisión. Cada etapa se entrena utilizando una técnica llamada refuerzo, el cual proporciona la capacidad de entrenar a un clasificador altamente preciso tomando un promedio ponderado de las decisiones tomadas por los aprendices débiles.

Cada etapa del clasificador etiqueta la región definida por la ubicación actual de la ventana deslizante como positiva o negativa. Ser positiva indica que se encontró un objeto y ser negativa indica que no se encontraron objetos. Si la etiqueta es negativa, la clasificación de esta región está completa y el detector desliza la ventana a la siguiente ubicación. Si la etiqueta es positiva, el clasificador pasa la región a la siguiente etapa. El detector informa sobre un objeto encontrado en la ubicación de la ventana actual cuando la etapa final clasifica la región como positiva.

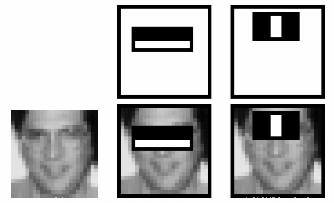


Figura 9: Uso de ventanas deslizantes para clasificación de rostros

Las etapas están diseñadas para rechazar muestras negativas lo más rápido posible. El supuesto es que la gran mayoría de las ventanas no contienen el objeto de interés. Por el contrario, los verdaderos positivos son raros y vale la pena tomarse el tiempo para verificarlos.

* Un verdadero positivo ocurre cuando una muestra positiva está correctamente clasificada.
* Un falso positivo ocurre cuando una muestra negativa se clasifica erróneamente como positiva.
* Un falso negativo ocurre cuando una muestra positiva se clasifica erróneamente como negativa.

Para funcionar bien, cada etapa de la cascada debe tener una tasa baja de falsos negativos. Si una etapa etiqueta incorrectamente un objeto como negativo, la clasificación se detiene y no puede corregir el error. Sin embargo, cada etapa puede tener una alta tasa de falsos positivos. Incluso si el detector marca incorrectamente un objeto no positivo, puede corregir el error en etapas posteriores. Agregar más etapas reduce la tasa general de falsos positivos, pero también reduce la tasa general de positivos verdaderos. [9]

#### Análisis de componente principal

El análisis de componentes principales (PCA) es una técnica utilizada para describir un conjunto de datos en términos de nuevas variables, o componentes, no correlacionadas. Las componentes se ordenan por la cantidad de varianza original que describen, por lo que la técnica es útil para reducir la dimensionalidad de un conjunto de datos.

El PCA busca la proyección según la cual los datos queden mejor representados en términos de mínimos cuadrados. Esta convierte un conjunto de observaciones de variables posiblemente correlacionadas en un conjunto de valores de variables sin correlación lineal llamadas componentes principales [10], cuyo número será igual o menor a las componentes iniciales.

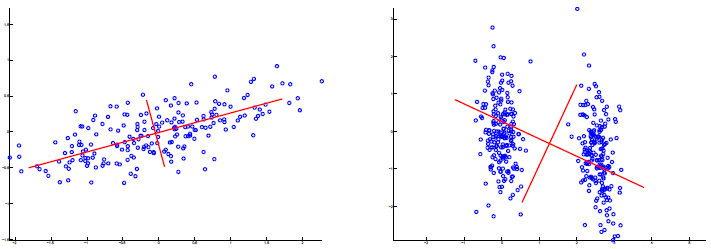


Figura 10: Ejemplo del uso de PCA para clasificación de muestras

Su uso en la clasificación de muestras radica en su capacidad para reducir la dimensionalidad de las muestras sin perder información relevante, lo cual presenta una ventaja notable al compararse con la técnica de reducción de resolución de imagen utilizada normalmente. Sin embargo, presenta la desventaja de que necesita una gran cantidad de muestras para ser utilizada, y no puede aplicarse a una imagen individual obtenida en vivo.

### Lenguaje de programación Python

Python es un lenguaje de programación interpretado a código abierto, cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, de tipado fuerte y dinámico y multiplataforma. [11]

Este lenguaje cuenta con un importante número de librerías tanto para trabajar con redes neuronales artificiales como para aplicar técnicas de visión artificial. Es además un lenguaje ampliamente utilizado y con documentación completa y accesible.

#### Librerías Tensorflow y Keras

Las librerías Tensorflow y Keras son las más utilizadas en el campo de la investigación para trabajar con *Machine Learning* en general, por contar con funciones y métodos especializados para trabajar con redes neuronales artificiales.

Tensorflow fue desarrollada por Google específicamente para *Machine Learning*. Su unidad de procesamiento es el Tensor, el cual es una construcción específica similar a un vector, y es la base arquitectónica de este lenguaje.

Keras, por su parte, es una librería basada en Tensorflow y que permite programar y entrenar redes neuronales de varios tipos en pocas líneas de código. De esta forma, una vez decidida su arquitectura, no es necesaria su construcción capa por capa, sino que se debe llamar a la función correspondiente, introducir los parámetros necesarios, y utilizar métodos para entrenar la redes o probar sus resultados.

Esta librería cuenta con todos los métodos presentes en Tensorflow para el desarrollo de redes neuronales, entre los que destacan reductores de medias, optimizadores, convolucionadores, funciones de *max-pooling*, entrenadores de redes, calculadores de eficacia de la red, entre otros. Sin embargo, incluye funciones prefabricadas que permiten su utilización de una forma más simple y rápida.

#### OpenCV

La librería OpenCV es la librería más amplia y versátil que se puede encontrar en este lenguaje para el procesamiento de imágenes. Cuenta con funciones y métodos de fácil uso que permiten realizar una amplia serie de acciones sobre archivos de imágenes: recortes, transformaciones de color, erosión, dilatación, seguimiento de objetos, tratamiento de ruido, entre otros.

Para lograr esto, Python aprovecha la interpretación de una imagen como una matriz que hace el computador, para realizar distintas operaciones matriciales sobre esta y realizar su procesamiento.

OpenCV cuenta con todas las operaciones necesarias para tratar cualquier imagen de forma secuencial y extraer la información que sea necesaria de esta, de forma tal que puede ser utilizada para el tratamiento en vivo de imágenes en vídeo.

### Proyectos similares

En el pasado reciente, ha habido un interés cada vez mayor en el uso de tecnologías como las mencionadas anteriormente para la clasificación de imágenes de todo tipo. Las técnicas utilizadas en cada uno de ellos varían dependiendo de las necesidades del problema, pero tienen en común el procesamiento previo de imágenes y su posterior clasificación mediante redes neuronales.

#### Detecting Skin in Images & Video Using Python and OpenCV

Este trabajo, publicado abiertamente por Adrian Rosebrock [12], consiste en la elaboración de un algoritmo capaz de detectar y filtrar la piel humana en un vídeo en vivo mediante el uso de la librería OpenCV para visión artificial, en lenguaje Python.

Utilizando la librería OpenCV, el autor transforma la imagen a composición de color HSV y luego define un rango de colores dentro del cual considera que está la piel humana. El rango de colores utilizados en HSV varían entre:

* [0,48,80]:
* [20,255,255]:

A partir de este punto, todos los píxeles de la imagen que estén dentro de estos valores se consideran piel, por lo que se pasa a extraer de la imagen original una máscara binaria (blanco y negro), la cual es utilizada como filtro sobre la misma imagen original, en una operación que sólo permite píxeles dentro de dicho umbral.

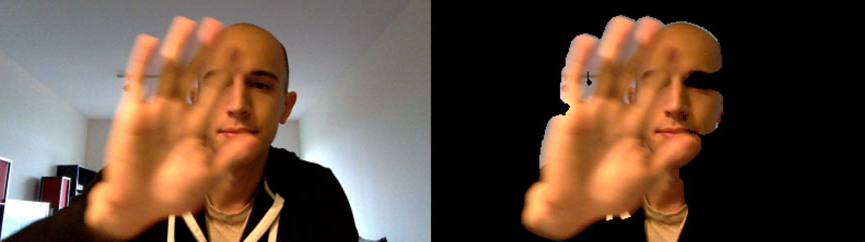


Figura 11: Detección de piel y filtrado en imagen

Finalmente, se presenta en secuencia de vídeo la imagen filtrada, donde es posible ver que sólo la piel es mostrada.

#### Clasificación de número escritos a mano con redes neuronales

Desde hace varios años, este se ha vuelto un problema clásico en el estudio de las redes neuronales. Fue planteado utilizando la base de datos modificada Instituto Nacional de Estándares y Tecnología (MNIST, en inglés), que consiste en una base de datos indexada de decenas de miles de muestras de dígitos escritos a mano provenientes de archivos civiles de EEUU. Está normalizada como imágenes en escala de grises y resolución de 28x28 píxeles.

En un trabajo anterior, se procedió a analizar la efectividad de diferentes clasificadores sobre sobre esta base de datos para lograr identificar los dígitos escritos a mano. Se utilizaron 10.000 muestras para esto, de las cuales el 80% se utilizó para entrenamiento y el 20% restante para pruebas.

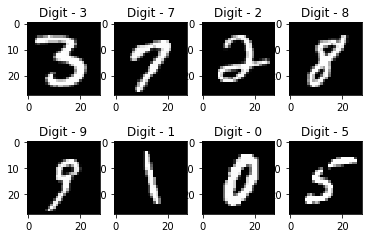


Figura 12: Ejemplo de dígitos de la base de datos MNIST

Para reducir el tiempo de procesamiento y optimizar el proceso de clasificación, se procedió primero a identificar píxeles que no estuvieran aportando información (principalmente en las esquinas). Para ello se revisó el valor de cada uno de los píxeles para las 10.000 imágenes, y aquellos que tuvieran exactamente el mismo valor en todas ellas, se eliminó. Este proceso redujo la dimensionalidad en un 10%.

Posteriormente se aplicó un análisis de componente principal (PCA, en inglés) para reducir aún más la dimensionales perdiendo la menor información posible. El número de dimensiones utilizadas dependió del clasificador que se estudió en cada caso.

Los resultados obtenidos fueron los siguientes:

* Clasificador KNN: es un método de clasificación supervisado, que estima la probabilidad de un elemento de pertenecer a una clase dependiendo de la clase a la que pertenezca el K número de vecinos. El porcentaje de acierto para este clasificador fue del 94,55%.
* Clasificador bayesiano: es un clasificador probabilístico que aplica el Teorema de Bayer y los supuestos de independencia fuertes (ingenuos) entre las clases para determinar a dónde pertenece cada muestra. En lugar de crear un límite en función de la posición de cada muestra individual, este clasificador crea el límite en función de la distribución probabilística de cada conjunto (clase) de muestras. El porcentaje de acierto para este método fue del 85,65%.
* Clasificador MPL: este tipo de red neuronal se entrena y ajusta sus ponderaciones para reproducir el mismo resultado con entradas similares. Se decidió utilizar un PCA para reducir los datos hasta 100 dimensiones, y el resultado final fue del 93,05%.
* Clasificador SOM: esta red neuronal se comporta como un mapa que trata de cubrir la mayoría de las muestras de datos, de modo que a cada muestra se le asigna a una neurona. Se utilizó una topología de 60x7 neuronas y un PCA para reducir la dimensionalidad a 200. El porcentaje de acierto para este clasificador fue del 73,75%.

#### Sign Language Recognition with Unsupervised Feature Learning

Este trabajo fue publicadopor Justin Chen en la Universidad de Stanford (EE.UU.) en 2014 [13], y trata el reconocimiento del lenguaje de señas americano para sordomudos utilizando redes neuronales no supervisadas y procesamiento de imágenes. Está inspirado en el trabajo sobre clasificación de números escritos a manos, y utiliza una dataset de 1200 muestras para cada una de las señas de las 10 primeras letras del alfabeto latino, omitiendo la ”J” al esta requerir movimiento.

Para su procesamiento, se experimentó utilizando diferentes aproximaciones:

* Segmentación de bordes: esta estrategia pretendía utilizar la función cv2.canny() de OpenCV, la cual consiste en un detector de bordes en la imagen. Para ello, realiza derivaciones tanto vertical como horizontalmente, de manera que identifique los cambios repentinos en la imagen como bordes. Una vez teniendo esta imagen de bordes, se procede a la clasificación.
* Identificación de puntas de los dedos: la estrategia a seguir en este caso es utilizar técnicas geométricas para definir la posición aproximada de las puntas de los dedos. Para ello, se identifica la mano como una curva geométrica cerrada, y se procede a identificar los vértices de esta mediante el método cv2.contour() de OpenCV. Utilizando el número de dedos identificados, así como la posición relativa de estos con respecto al centro de masa de la imagen, se procede a la etapa de clasificación.
* Segmentación por color de piel: esta última estrategia hace una conversión de la imagen a blanco y negro, utilizando como umbral para cada píxel el hecho de si pertenece a el rango definido como piel humana. Posteriormente, se aplica esta imagen en blanco y negro como una máscara en la imagen principal, se forma tal que sea posible segmentar la mano.

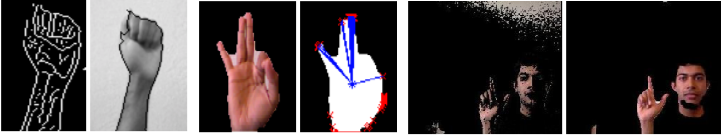


Figura 13: Métodos de procesamiento de imagen (de izquierda a derecha: segmentación de bordes, identificación de puntas de los dedos y finalmente, segmentación por color de piel)

Una vez detectada la mano, se procedió a recortarla y pasarla a una resolución de 32x32 píxeles, en escala de grises.

Para el entrenamiento, se utilizó un dataset de 1.200 imágenes por cada letra, para un total de 12.000 muestras. Cada muestra tiene una resolución de 32x32 píxeles. Por esta razón la topología utilizada fue de 32x32 neuronas (una por cada píxel) en la capa de entrada, 100 neuronas en la única capa oculta, y 10 neuronas en la capa de salida (una por cada letra). El número de ciclos para el entrenamiento se definió en 400.

De las tres metodologías utilizadas, la que demostró un mejor rendimiento fue el uso de la segmentación por color de piel. Los resultados finales lograron superar el 95% de aciertos para algunas letras, lo cual es comparable con los resultados del clasificador de dígitos escritos a mano.

# METODOLOGÍA Y DESARROLLO

En esta sección se explicará con detalle la metodología utilizada para resolver la problemática planteada y los retos que surgieron durante la elaboración de este proyecto, así como el procedimiento que se ha seguido durante el desarrollo.

### Elección del lenguaje de programación

Como se mencionó en el capítulo anterior, el lenguaje de programación seleccionado para el desarrollo de este proyecto fue Python, más específicamente, en su versión 3.6.

Las razones de esta decisión se pueden resumir en dos puntos principales:

* Python es el lenguaje de programación más ampliamente difundido en la actualidad para proyectos basados en inteligencia artificial, más específicamente, *Machine Learning*. Existe una amplia variedad de cursos gratuitos sobre su aplicación en *Machine Learning*, así como innumerables trabajos publicados que pueden servir de guía a la hora de buscar estrategias alternativas.
* Tanto el robot Pepper como el robot Nao tienen entre sus posibilidades el llamado de servicios en la nube que estén desarrollados en Python o C#.

Si bien el primer punto tuvo un gran peso en la decisión final, fue el segundo el que descartó por completo a MatLab como una de las posibilidades para este proyecto, a pesar de ser la herramienta utilizada a lo largo del Máster y en la que se tenía una mayor experiencia.

Por otra parte, la elección de la versión 3.6 de Python por encima de la más difundida (v2.7) o la más reciente (v3.7) se debió al hecho de que todas las librerías que serían necesarias en este proyecto sólo estaban disponibles simultáneamente en esta versión al momento de iniciar el proyecto, además de tener con una mayor robustez al contar con más tiempo desde su lanzamiento al público

#### Librerías utilizadas

La siguiente lista enumera todas las librerías que son utilizadas durante la ejecución de los programas desarrollados en este proyecto:

* Numpy
* OpenCV
* OpenCV-Contrib
* OS
* Shutil
* Math
* Scipy
* Skimage
* PIL
* Matplotlib
* Tensorflow
* Keras
* Sklearn
* Cmake
* Dlib

La instalación de las librerías mencionadas, luego de la instalación de la versión 3.6 de Python, es necesaria para la ejecución de los programas. Para ello, es recomendable instalar previamente el manejador de paquetes de Python **pip**, el cual se puede encontrar gratuitamente en línea.

El orden de instalación recomendado es el orden de la lista anterior, y para realizar la instalación de cada una, se debe contar con conexión a Internet y abrir la ventana de comandos de Windows (CMD) como administrador para escribir la siguiente línea:

**pip install** *(nombre de la librería)*

Luego, presionar *Enter* y esperar a que se complete antes de instalar la siguiente. Esta acción se debe repetir para todas las librerías.

Finalmente, para poder instalar las dos últimas librerías, *Cmake* y *Dlib*, es necesario instalar el módulo “Visual C++ Tools for Cmake” de Visual Studio 2015 o superior.

### Obtención de bases de datos para entrenamiento de redes neuronales

Para el entrenamiento de las redes neuronales artificiales necesarias para la resolución de ambos problemas, se consideró primeramente la generación de las bases de datos utilizando una colección de fotografías creadas específicamente para el proyecto. Sin embargo, para alcanzar un número de fotografías tal que permitieran resultados aceptables, este trabajo tomaría mucho tiempo, por lo que se consideró el uso de bases de datos ya existentes y disponibles de forma gratuita.

#### Base de datos de rostros expresando emociones

La Universidad de Columbia, en EE.UU., cuenta con una base de datos de 672 fotografías de 81 modelos, con una resolución de 506x650 píxeles en formato PNG que fueron recopiladas por la Dra. Nim Tottenham. Estas fotografías muestran el rostro de diferentes modelos expresando una serie emociones, tanto con la boca abierta como con la boca cerrada. Las expresiones mostradas por cada persona son:

* Neutral o calmado
* Alegría
* Tristeza
* Ira
* Miedo
* Desagrado
* Sorpresa

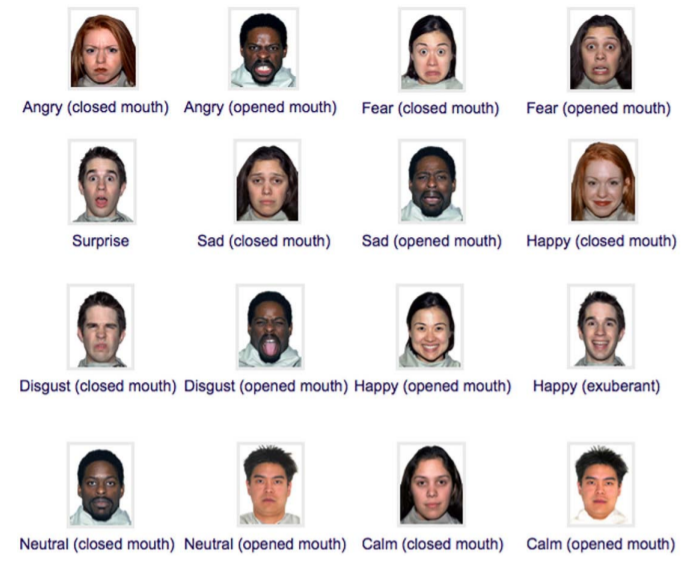


Figura 14: Muestras del dataset utilizado para identificación de emociones

Las expresiones faciales mostradas por los modelos son actuadas y exageradas, por lo que, a pesar de ser claramente identificables, se nota que no son naturales. Cada uno de los modelos mostró varias expresiones faciales, tanto con la boca abierta como con la boca cerrada. La etiqueta para la expresión mostrada en la fotografía forma parte del nombre del archivo de imagen.

La composición demográfica del conjunto de 81 modelos es la siguiente:

* Participantes de origen caucásico: 58 (71,6%)
* Participantes de origen africano: 5 (6,2%)
* Participantes de origen latino: 4 (4,9%)
* Participantes de origen asiático: 4 (4,9%).

Además, 51 de los modelos son mujeres (62,9%) y 30 son hombres (37,1%), con una media de edad total de 23,4 años.

El dataset se dividió aleatoriamente, de forma tal que 538 fotografías (80%) se utilizaron para el entrenamiento y 134 fotografías (20%) se utilizaron para pruebas.

#### Base de datos de manos haciendo señas del lenguaje de sordomudos

El mismo instituto mencionado en el segundo capítulo como propietario de la base de datos MNIST, la cual es utilizada para el entrenamiento de programas para el reconocimiento de dígitos escritos a mano, también tiene a su disposición una base de datos con 34.627 fotografías de manos en las distintas posiciones del lenguaje de sordomudo americano, con la excepción de la “J” y la “Z”, al no ser estáticas.

Esta base de datos se encuentra publicada abiertamente, y forma parte de uno de los retos de inteligencia artificial en *Kaggle*, una popular comunidad en línea de científicos de datos y aprendices de máquinas, propiedad de Google LLC, con más de un millón de usuarios registrados actualmente.

El objetivo principal de esta base de datos es que sea utilizada para la elaboración de proyectos de clasificación de imágenes individuales, las cuales contengan manos haciendo señas del lenguaje de sordomudos, razón por la cual no cuenta con muestras para las letras “J” o “Z”, ya que estas requieren movimiento.

Puesto que en este trabajo tampoco se está considerando la identificación de dichas letras, se decidió utilizar esta base de datos aplicada a la clasificación de las mismas señas, pero en vídeos en vivo.



Figura 15: Alfabeto contenido en el dataset

Cada una de estas imágenes se encuentra en resolución de 28x28 píxeles y en escala de grises de 256 bits. Además, se encuentran codificadas en un archivo CSV y en forma vectorial en lugar de matricial. Es decir, cada una de las imágenes tiene todos sus píxeles en una sola línea, y todas las líneas comienzan por un primer valor, que es la etiqueta (donde “0” corresponde a “A”, “1” corresponde a “B”, “2” corresponde a “C”, y así sucesivamente, hasta el “23” que corresponde a “Y”).

El total, incluyendo la etiqueta, es de 785 celdas por línea, y el total de líneas es de 34.627, divididas en dos archivos: uno de 27.455 (80%) para entrenamiento, y otro de 7.172 (20%) para pruebas.

### Preprocesamiento del dataset

Con el objetivo de mejorar el rendimiento del clasificador, se procedió a realizar una serie de modificaciones a las imágenes originales del dataset, utilizando las distintas herramientas que ofrece la librería OpenCV de Python.

#### Preprocesamiento de imágenes de rostros

Para realizar el preprocesamiento de cada una de las imágenes de rostros, se utilizó el procedimiento mostrado en el siguiente flujograma y explicado a continuación:

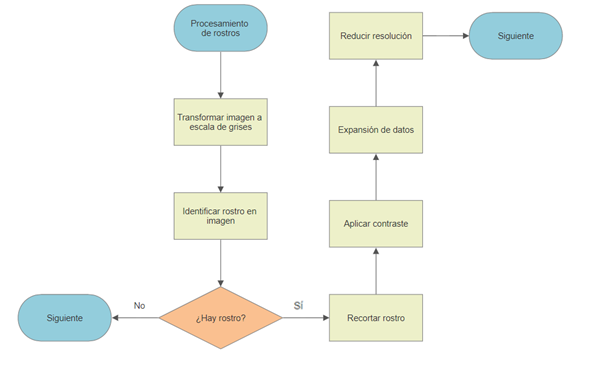


Figura 16: Diagrama de flujo de procesamiento de imágenes de rostros

El primer paso que se realiza una vez cargadas las imágenes es transformarlas de RGB a escala de grises. Para ello, se utiliza la siguiente función:

cv2.cvtColor(**image**, cv2.COLOR\_BGR2GRAY)

Luego se utiliza el clasificador de LPE Cascade (incluido en la librería OpenCV) para identificar la zona de interés exacta donde se encuentra el rostro dentro de la imagen, y se recorta esta de manera tal que se minimice el número de píxeles que forman parte del fondo, cuello, ropa y cabello del modelo, dejando únicamente el rostro, y disminuyendo el potencial ruido. Para realizar esta extracción sistemática de los rostros, se definió la siguiente función:

**def** detect\_faces(**imagen**):

**face\_detector** = dlib.get\_frontal\_face\_detector()

**faces** = **face\_detector**(**imagen**, 1)

**crop**=[(**x**.left(),**x**.top(),**x**.right(),**x**.bottom()) **for** **x** **in** **faces**]

**return** **crop**

Con la intención de mejorar calidad de los detalles de los rostros, se decidió definir una función capaz de aumentar el contraste de la imagen. OpenCV no cuenta directamente con una función capaz de hacer esto, sin embargo, se utilizó una serie de funciones que en conjunto producen este mismo efecto. Dicha función se muestra a continuación:

**def** contraste(**imagen**):

**clahe** = cv2.createCLAHE(*clipLimit*=3., *tileGridSize*=(8, 8))

**lab** = cv2.cvtColor(**imagen**, cv2.COLOR\_BGR2LAB)

**l**, **a**, **b** = cv2.split(**lab**)

**l2** = **clahe**.apply(l)

**lab** = cv2.merge((**l2**, **a**, **b**))

**contraste** = cv2.cvtColor(**lab**, cv2.COLOR\_LAB2BGR)

**return** **contraste**

Al tratarse de un dataset relativamente pequeño, se corre el riesgo de que la solución no responda correctamente ante mínimos cambios. Es por ello por lo que se ha utilizado una función de expansión de datos, la cual consiste en replicar cada una de las imágenes haciendo ciertas modificaciones sobre ella.

Para realizar esta expansión de datos, se utilizó principalmente una función de la librería Keras para preprocesamiento de imágenes llamada ImageDataGenerator()[14]. Dicha función cuenta con hasta 22 diferentes parámetros para la expansión de datos.

Los parámetros utilizados fueron:

* Rotación aleatoria de imágenes entre -15º y 15º.
* Introducción de ruido gaussiano.
* Inversión horizontal de la imagen.
* Modelo de rellenado: “*nearest*” (completar vacíos con píxeles de color similares a los píxeles vecinos)

Al hacer esto, el dataset original de 672 imágenes pasó a tener 10 veces más datos de muestra.

Por último, se procede a reducir la resolución de las imágenes a 32x32 píxeles. Para realizar esta transformación, se utilizó la siguiente función:

cv2.resize(**imagen**, (32, 32))

Esto se hace puesto que, si se utilizara el dataset original, la capacidad de procesamiento necesaria para entrenar una red neuronal de tales dimensiones no está disponible en un ordenador convencional.

Utilizando como base la propuesta del trabajo presentado en el punto 3.4.3, y luego de realizar varias pruebas, se concluyó que esta dimensión fue la que mejor proporción entre resolución e información produjo.

#### Preprocesamiento de imágenes de señas

De igual forma que en el caso anterior, es necesario hacer modificaciones a las imágenes originales del dataset para obtener los mejores resultados.

En este caso, las imágenes ya se encuentran en escala de grises de 256 bits y una resolución de 28x28, por lo que pueden ser utilizadas en un ordenador convencional.

Sin embargo, se aplicó nuevamente la expansión de datos, utilizando nuevamente los mismos parámetros que en el punto anterior para mejor la calidad de estos.

Cabe destacar que la inversión horizontal en este caso es muy importante, ya que todas las fotografías son de manos diestras, y puesto que se espera que el programa resultante sea capaz de interpretar a personas diestras y zurdas por igual, es necesario tener también la versión de cada una de las señas en una mano zurda.

### Entrenamiento de redes neuronales artificiales

El proceso de entrenamiento de las redes neuronales artificiales para cada uno de los dos problemas fue similar, siendo la principal diferencia entre ambas el número de neuronas de entrada y el número de neuronas de salida, debido a que tanto la resolución de las imágenes de entrada como el número de posibles resultados varía en cada caso. Esta diferencia modificó la arquitectura de la red neuronal.

La librería Keras de Python cuenta con el método train(), el cual permite entrenar una red neuronal definida previamente, únicamente necesitando una serie de parámetros, consistentes en: el dataset de entrada, las etiquetas de dicho dataset, la red neuronal propiamente dicha, el número de ciclos, el tamaño de los lotes, y la función de entropía.

#### Diseños previos de clasificadores

Antes de lograr alcanzar el diseño actual de los clasificadores, se hicieron varios intentos infructuosos. Todos ellos buscaban conseguir una mejora sobre el funcionamiento del algoritmo, pero por una u otra razón, fueron dejadas de lado o tuvieron que ser mejoradas drásticamente a través de otra estrategia.

Las diferentes aproximaciones que fueron utilizadas para el diseño y entrenamiento de las redes neuronales, y que posteriormente fueron descartadas, se indican a continuación:

* Programación manual de la red neuronal: El primer intento de utilizar una red neuronal en Python se realizó de forma manual, utilizando la librería Tensorflow y sin la utilización de la librería Keras. Para ello se definieron manualmente funciones de convolución, *max-pooling*, cálculo de precisión de la red, variables de pesos entre neuronas y variables de *bias*. De igual manera, se diseñó manualmente la función de entrenamiento, con un tope de 3.000 iteraciones o un máximo de un 95% de precisión para evitar el sobreentrenamiento. A pesar de mostrar resultados similares al modelo finalmente utilizado, en la práctica tenían un muy bajo rendimiento en la clasificación en vivo por su baja velocidad de procesamiento.
* Aplicación de análisis de componentes principales (PCA): En un intento por reducir la dimensionalidad de las muestras, se intentó utilizar PCA. El problema principal es que el tamaño original de las imágenes (506x650 píxeles) es muy grande para ser procesado en vivo. Se comprobó que un tamaño más reducido permitía mejorar el rendimiento computacional, por lo que se decidió aplicar PCA. Sin embargo, al realizar la comparación de los resultados entre la red entrenada con un dataset redimensionado utilizando PCA, y otro utilizando escalamiento simple, demostró que la segunda opción a pesar de ser más sencilla fue más efectiva. Adicionalmente, la red neuronal resultante solo funcionaría con imágenes redimensionadas utilizando PCA, lo cual dificultaría su utilización en la clasificación en vivo, donde se va obteniendo una imagen a la vez.
* Inversión manual de imagen: Antes de conocer la función de expansión de datos de la librería Keras, se invirtió manualmente cada imagen para duplicar el dataset. Los resultados fueron positivos, pero se decidió utilizar la versión de Keras al ser más versátil, simple y con más opciones.
* Segmentación del objeto principal: Con objetivo de eliminar la mayor cantidad de ruido dentro de la imagen proveniente del fondo de esta, se consideró aislar el objeto principal (rostro o mano, dependiendo del caso) utilizando un filtro de movimiento. La idea principal era utilizar el hecho de que el fondo es estático para utilizar un filtro de movimiento que permitiera sustraerlo de la imagen, dejando solo a la persona. Lamentablemente, no se fue posible perfeccionar esta técnica, principalmente porque a pesar de que la persona se mueve, grandes porciones de píxeles del rostro o la mano se mantenían lo suficientemente invariables para que comenzaran a ser descartados por el filtro, dejando agujeros importantes. Adicionalmente, cualquier cambio en la iluminación o foco de la cámara afectaba al filtro.
* Utilización de siluetas o bordes: Puesto que se estaba filtrando el objeto principal por color, para el clasificador de señas se probó utilizar una red neuronal entrenada con imágenes que solo tuvieran la silueta de la mano (utilizando la propia máscara obtenida en el preprocesamiento) o los bordes (utilizando la función cv2.canny() de OpenCV). En ambos casos los resultados obtenidos no presentaron ningún tipo de mejora en la clasificación.

Cabe destacar que el tratamiento de imágenes que se utilizó en estas estrategias impactaba también el utilizado en la clasificación en vivo, por lo que las estrategias aquí mencionadas fueron probadas en ambas partes del algoritmo.

Una vez que fueron descartadas las estrategias anteriores, se decantó por el uso de aquellas que se describen a continuación para cada uno de los clasificadores.

#### Entrenamiento del clasificador de expresiones faciales

Ya teniendo las imágenes que serán utilizadas para el entrenamiento de la red neuronal y habiendo sido preprocesadas, se procedió a realizar su ordenamiento, codificación, separación del dataset, entrenamiento y pruebas posteriores. Este procedimiento se muestra a continuación, y detallado posteriormente:

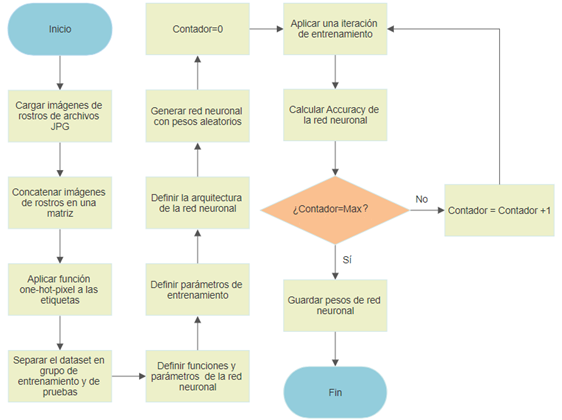


Figura 17: Diagrama de flujo del entrenamiento del clasificador de expresiones faciales

Para comenzar, se procede a cargar secuencialmente una a una cada imagen de los archivos con formato JPG desde la ruta donde se encuentran almacenado. Durante este paso, se aplica individualmente el preprocesamiento indicado en el punto 4.3.1 de este documento.

Una vez que la imagen se ha transformado a escala de grises con resolución de 32x32 píxeles, se pasa a aplicar la siguiente función de la librería Numpy:

np.concatenate(**imagen**)

Entendiendo que, computacionalmente hablando, una imagen no es más que una matriz, esta función se está encargando de “autoconcatenar” dicha matriz. Es decir, está convirtiendo una matriz mxn en un vector de largo m veces n:

Cada una de estas imágenes autoconcatenadas se comienza a almacenar una debajo de otra en una nueva matriz. Dicha matriz será la matriz de entrada de la red neuronal.

De forma paralela, por cada imagen que se guarda en dicho dataset, se obtiene del nombre del archivo cuál es la etiqueta de dicha imagen (la cual se corresponde con la expresión facial mostrada en la misma) y se va guardando en un vector, cuyo orden se corresponde con la matriz del dataset de entrada. Cada una de estas etiquetas se almacena con un valor numérico de la siguiente manera:

1. Ira
2. Neutral
3. Disgusto
4. Miedo
5. Alegría
6. Tristeza
7. Sorpresa

Una vez que todas las imágenes han sido transformadas y guardadas, se procede a aplicar la función de Keras OneHotEncoder(). Esta función aplica una codificación “*One Hot Pixel*”, la cual consiste en transformar cada una de las etiquetas a un vector de un largo igual al número de diferentes clases de etiquetas, donde todos los valores sean cero excepto por uno, que se corresponde con el de la etiqueta correspondiente, el cual tiene un valor igual a uno:

Luego de esta transformación, se tiene que el dataset de entrada es una matriz de largo 1024 (32x32) y de alto 672 (número de imágenes), mientras que el dataset de etiquetas tiene un largo de 7 y el mismo alto de 672.

A continuación, se procede a dividir el dataset en dos subgrupos: entrenamiento y pruebas. Para ello, se divide de forma aleatoria utilizando la siguiente función de Keras:

train\_test\_split(**x**, **y**, *test\_size*=0.2)

Donde **x** es el dataset de entrada, **y** es el dataset de etiquetas, y 0.2 corresponde con el porcentaje del dataset que se usará en pruebas. Es decir, el dataset original se divide en un 80% para entrenamiento, y un 20% para pruebas.

El siguiente paso consiste en definir los distintos parámetros y funciones que serán utilizados para diseñar la arquitectura de la red neuronal. Las funciones que se utilizarán en este caso son la función de convolución, la función de activación y la función de *max-pooling*.

Como se mencionó en el capítulo anterior, la función de convolución es la principal característica de una red neuronal convolucional. Dependiendo del tipo de problema o del tamaño de la imagen a clasificar, el tamaño de la matriz de convolución es muy importante.

Gracias a la versatilidad de la librería Keras, no es necesario escribir cada una de las funciones mencionadas, sino que solo se necesita definir los parámetros que estas utilizarán. Esto permite utilizar un método que agrega dicha función a las capas ocultas de la red neuronal que se está construyendo, y se muestra a continuación:

**classifier**.add(

Convolution2D(16, (3, 3),*activation*="relu",*input\_shape*=**x**\_train.shape[1:])

)

En dicha función se pueden observar los siguientes parámetros:

* Número de filtros: Se trata de los filtros utilizados a lo largo de las capas de la red neuronal. Se elige la cantidad de filtros en función de la complejidad de la tarea. Tareas más complejas requieren más filtros, y generalmente el número de filtros crece después de cada capa. Las primeras capas (con menor número de filtros) detectan algunas de las características simples de las imágenes (bordes, tono de color, etc.) y las siguientes capas intentan obtener características más complejas basadas en las más simples. El número de filtros seleccionado para esta red fue **16**.
* Tamaño de la matriz de convolución: Esta matriz se encarga de extraer las características de la imagen que activarán o no la neurona en la siguiente capa de la red. Por lo general, los tamaños oscilan entre 2x2 y 5x5 píxeles, y en este caso, se seleccionó de forma empírica un tamaño de **3x3** al tratarse de imágenes de 32x32 píxeles.
* Forma del dataset de entrada: Está relacionado directamente con la morfología de la primera capa de la red neuronal. En este caso tendrá un tamaño igual a **1024** (32x32).
* Función de activación: Es la función utilizada para decidir si un estímulo va a pasar de una capa de la red neuronal a la siguiente. Como se mencionó en el capítulo anterior, para las redes neuronales convolucionales se utiliza una función tipo capa de rectificación, o RELU (en inglés), para las capas ocultas. Existe la posibilidad de utilizar la RELU clásica o suavizada, y en este caso en particular, se decidió utilizar la clásica. Se puede observar una comparación de ambas funciones a continuación:

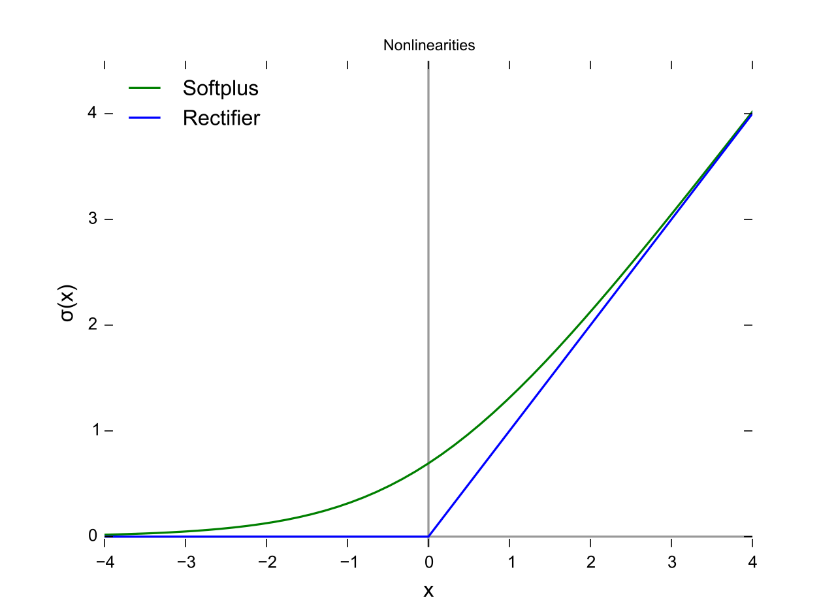


Figura 18: Gráfica de la función RELU clásica (azul) y suavizada (verde)

La función de *max-pooling* también está predefinida en la Librería Keras. Se decidió utilizar una matriz de 2x2 para ello:

**classifier**.add(MaxPooling2D(*pool\_size* =(2,2)))

Finalmente, se agregó la capa de salida de la red neuronal, la cual consiste en un perceptrón simple que hace la función de neuronas de clasificación. El tamaño de esta capa es igual al número de clases (7) y utiliza como función de activación una sigmoide:

**classifier**.add(Dense(*units*=**n\_classes**, *activation*="sigmoid"))

La gráfica de la función sigmoide utilizada para esta capa de clasificación se muestra a continuación:

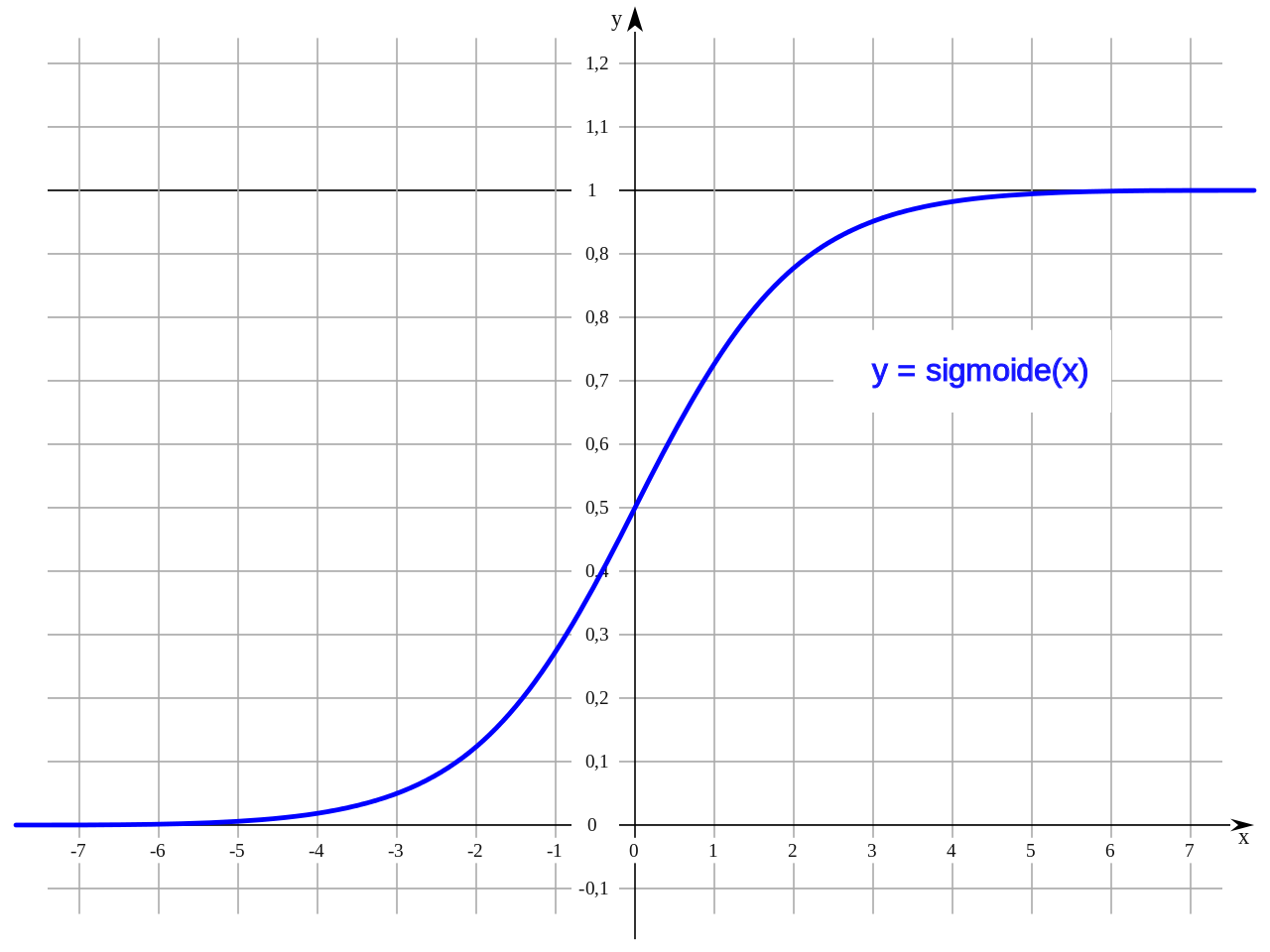


Figura 19: Gráfica de la función sigmoide

La librería Tensorflow incluye un método de optimización durante el entrenamiento de redes neuronales, siendo el más extendido en la actualidad el tipo “Adam”. De igual manera, el método de evaluación del error utilizado en este caso es “*Categorical Cross Entropy*”, el cual es el más recomendado para entrenamiento de CNN con más de dos clases. Por último, la métrica utilizada para evaluar la calidad de la red neuronal será el porcentaje de aciertos. Toda esta información se incluye a través del compilador del entrenador de la red neuronal, de la siguiente manera:

**classifier**.compile(

*optimizer*="adam",

*loss*="categorical\_crossentropy",

*metrics*=["accuracy"])

El tamaño del lote de muestras a ser utilizado en cada ciclo, así como el número de ciclos, se definió de forma empírica mediante prueba y error. Los valores finales que se utilizaron fueron lotes de 104 imágenes entrenando durante 100 ciclos.

Para realizar el entrenamiento de la red neuronal utilizando los datasets de entrada y etiqueta, se utilizó la siguiente función de la librería Keras:

**classifier**.fit\_generator(

**datagen**.flow(**x\_train**, **y\_train**, *batch\_size*=**batch**), *steps\_per\_epoch*= **x\_train**.shape[0]//**batch**,

*epochs*=**epochs**)

En dicha función se pueden observar los siguientes parámetros, en orden de aparición:

* Dataset de entrada.
* Dataset de etiqueta.
* Tamaño del lote.
* Número de pasos por lote (cociente entre el tamaño del dataset de entrada y el tamaño del lote).
* Número de ciclos.

Al ejecutarse dicha función, se entrenará una red neuronal generada de forma aleatoria y de manera iterativa por el número de ciclos indicado. Una vez el número de ciclos máximo haya sido alcanzado, la etapa de entrenamiento terminará y se procederá a probarla con el dataset que se dejó para validación. Una vez conocido el porcentaje de acierto de este, se guarda la red neuronal obtenida para su posterior uso.

#### Entrenamiento del clasificador señas

Teniendo el dataset con las señas del lenguaje de sordomudos guardado en formato CSV, se procedió a realizar su preparación, entrenamiento y pruebas posteriores. Este procedimiento se muestra a continuación, y detallado posteriormente:

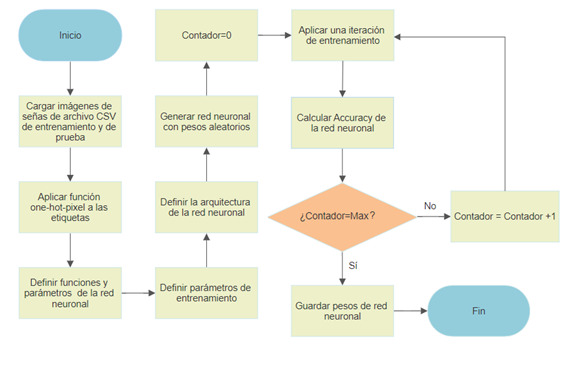


Figura 20: Diagrama de flujo del entrenamiento del clasificador de señas

El primer paso es cargar secuencialmente cada uno de los dos lotes de imágenes que se encuentran en los dos archivos CSV. Cada uno de estos archivos va a producir una matriz de la cual se obtendrá el dataset de entrenamiento y el de pruebas.

Cada una de estas matrices se separa en dos partes: un vector que corresponde con la primera celda de cada línea, y que es la etiqueta de cada imagen, y el resto de las celdas de cada línea que son las imágenes en sí.

Para cada una de las celdas del vector, el número se corresponde con alguna de las letras del abecedario de la siguiente manera:

1. A
2. B
3. C
4. D
5. E
6. F
7. G
8. H
9. I
10. K
11. L
12. M
13. N
14. O
15. P
16. Q
17. R
18. S
19. T
20. U
21. V
22. W
23. X
24. Y

A este vector se le procede a aplicar la función de Keras OneHotEncoder(). Esta función aplica una codificación “*One Hot Pixel*”, tal y como se explicó en la sección anterior, permitiendo transformar el vector de etiquetas en una matriz de un ancho igual al número posible de clases, y donde el único valor distinto a cero en cada línea es el valor posicional correspondiendo a la clase.

Luego de esta transformación, se tiene que, para el entrenamiento, el dataset de entrada es una matriz de largo 784 (28x28) y de alto 27.455 (número de imágenes), mientras que la matriz de etiquetas tiene un largo de 24 y el mismo alto de 27.455. Por otra parte, el dataset de pruebas es una matriz de 784 de largo y 7.172 de alto, con una matriz de etiquetas de 27 de largo y 7.172 de alto.

A continuación, se procede a definir el conjunto parámetros y funciones que serán utilizados para diseñar la arquitectura de la red neuronal. Al igual que en el clasificador anterior, las funciones que se utilizarán en este caso son la función de convolución, la función de activación y la función de *max-pooling*.

Se incluyó la función de convolución y de activación de las capas ocultas utilizando la misma función de la librería Keras que se explicó en el clasificador anterior. Sin embargo, en este caso el valor algunos de los parámetros cambiaron para adaptarse a la morfología de esta red:

* Número de filtros: El número seleccionado para esta red fue **16**, al igual que en el caso anterior.
* Tamaño de la matriz de convolución: Al tratarse de imágenes de un tamaño similar, también se utilizó un tamaño de **3x3** píxeles.
* Forma del dataset de entrada: En este caso se utilizó el tamaño de **784** (28x28).
* Función de activación: Nuevamente se utilizó la función de activación RELU clásica (ver figura 15).

La función de *max-pooling* utilizada es igual a la del caso anterior, con una matriz de 2x2. De igual manera, se agregó en la capa de salida de la red neuronal un perceptrón simple que haga la función de neuronas de clasificación. El tamaño de esta capa es igual al número de clases de este problema (24) y también utiliza como función de activación una sigmoide (ver figura 19).

El compilador del entrenador utilizado tiene exactamente las mismas características que el de la red neuronal explicada anteriormente.

Para este caso, el tamaño del lote de muestras a ser utilizado en cada ciclo, así como el número de ciclos, se definió nuevamente de forma empírica mediante prueba y error. Los valores finales que se utilizaron fueron 20 ciclos y lotes de 256 imágenes.

Para realizar el entrenamiento de la red neuronal usando los datasets de entrada y etiqueta, se utilizó la función **classifier**.fit\_generator() de la librería Keras. Los parámetros incluidos en el mismo son los equivalentes para este problema.

Al ejecutarse la función de entrenamiento, la red neuronal generada de forma aleatoria se entrena durante el número de ciclos indicado, como se explicó en el caso anterior. De igual manera, el entrenamiento se detendrá una vez el número de ciclos máximo haya sido alcanzado, y se probará con el dataset que se dejó para pruebas. Una vez conocido el porcentaje de acierto del mismo, se guarda la red neuronal obtenida para su posterior uso.

### Clasificación en vivo

El objetivo principal de este proyecto es la clasificación en vivo de las imágenes recibidas a través de una cámara para determinar la expresión facial mostrada en el rostro o la seña del lenguaje de sordomudos que se encuentren en ellas. Para ello, es necesario el desarrollo de un algoritmo capaz de realizar la adquisición y tratamiento de dichas imágenes, la identificación del objeto de interés y el seguimiento de este entre imágenes consecutivas, para posteriormente utilizar los clasificadores desarrollados con anterioridad, y que muestren la clasificación resultante en vivo.

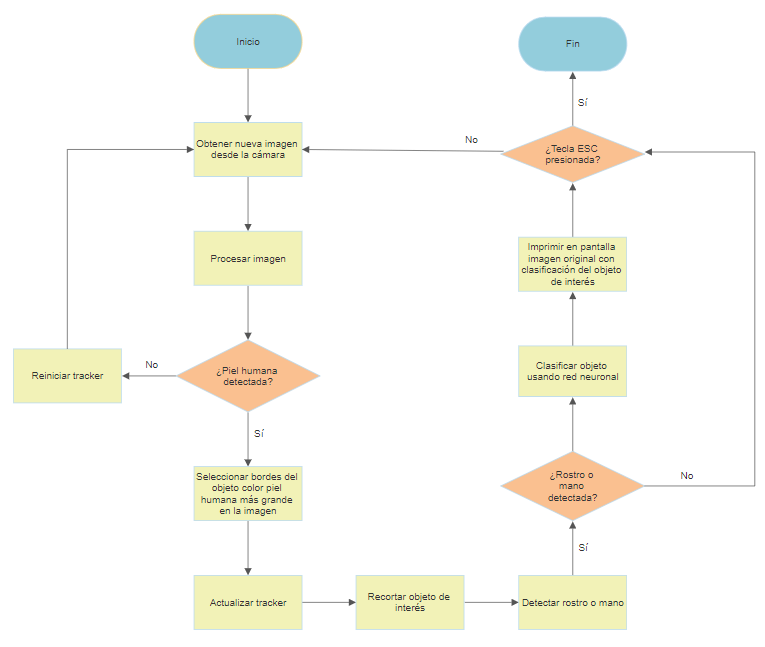


Figura 21: Diagrama de flujo del clasificador en vivo

La obtención de imágenes en vivo se realiza utilizando la cámara web integrada del ordenador, utilizando las herramientas ofrecidas por la librería OpenCV. Por cada uno de los cuadros de vídeo obtenidos, se aplica un algoritmo que prepara la imagen para ser clasificada, dependiendo de si se trata de la clasificación de expresiones faciales o de la clasificación de señas del lenguaje de sordomudos. En ambos casos el algoritmo desarrollado es muy similar, razón por la cual serán explicados como uno solo. Sin embargo, se hará especial énfasis en los puntos en los cuales no concuerden para permitir su diferenciación.

Para la adquisición de la imagen se está utilizando la cámara web integrada del ordenador portátil utilizado para ejecutar el algoritmo. Para acceder a sus imágenes, se utiliza la función cv2.VideoCapture(0) de la librería OpenCV. Esta función captura la imagen adquirida por la cámara al momento de ser llamada, y la presenta como una matriz con formato de color RGB. El dispositivo de entrada utilizado es seleccionado a través del argumento de la función, y se corresponde con el número de cámaras conectadas al equipo, empezando por cero.

Luego de la correcta adquisición de la imagen a través de la cámara, se procede a realizar el tratamiento de esta que permitirá identificar si existe un objeto con un color dentro del rango de la piel humana.

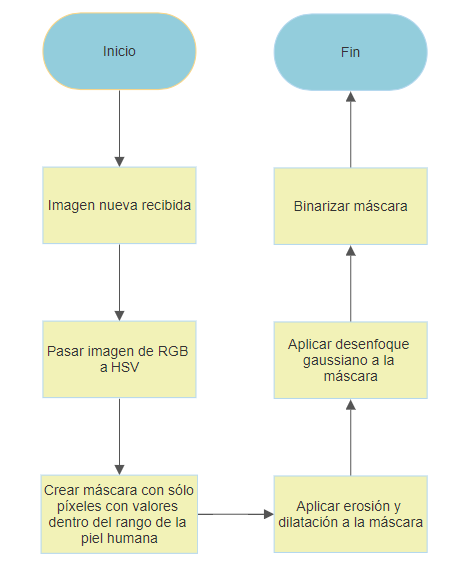


Figura 22: Diagrama de flujo del procesamiento en vivo de imágenes

Similar al preprocesamiento del dataset de entrenamiento de la red neuronal para clasificación de expresiones faciales, este algoritmo realiza una serie de operaciones sobre cada una de las imágenes recibidas para crear una máscara que sirva para detectar la posición de objetos con un color dentro del rango de la piel humana.

Para ello, primero realiza una transformación del modelo cromático RGB a HSV utilizando la siguiente función de la librería de OpenCV:

cv2.cvtColor(**Imagen**, cv2.COLOR\_BGR2HSV)

Esta transformación se hace siguiendo las especificaciones del trabajo de Adrian Rosebrock mencionado en el punto 3.4.1. Esta transformación permite definir con mayor facilidad el rango de la piel humana de la siguiente manera:

* Rango bajo: [0, 48, 80]:
* Rango alto: [20,255,255]:

El siguiente paso es utilizar dicho rango para definir una máscara. Una máscara es una matriz del mismo tamaño que la imagen original, compuesta de ceros y unos, la cual, al multiplicarse elemento por elemento con la imagen original, permite ocultar una parte de esta. En este caso, la máscara no se utilizará para ocultar o no partes de la imagen, sino para simplificar la información de la misma en la obtención de las coordenadas del objeto a analizar.

Con el rango de la piel humana ya establecido, se aplica la siguiente función de la librería OpenCV:

cv2.inRange(**Imagen**, **Rango\_Bajo**, **Rango\_Alto**)

Luego de aplicar esta función, se obtiene una imagen donde todos los píxeles que se encuentran fuera de dicho rango han sido sustituidos por ceros. En este punto, se procede a eliminar los conjuntos de píxeles que están dentro del rango de colores deseados, pero que se son muy pequeños para ser considerados objetos de interés, y que aportan ruido. Para ello, se aplica una erosión:

cv2.erode(**Imagen**, **Kernel**, *iterations*=2)

Inmediatamente se aplica también una dilatación. Hay dos principales objetivos para esta acción: primero, tratar de recuperar parte del “tamaño” de los objetos de interés que fueron erosionados, así como las conexiones que puedan pueden haberse perdido a causa de la erosión, sin correr el riesgo de recuperar el ruido puesto que estos objetos más pequeños fueron totalmente eliminados; segundo, completar cualquier agujero dentro de los objetos más grandes. Eso se logra mediante la siguiente función:

cv2.dilate(**Imagen**, **Kernel**, *iterations*=2)

La variable **Kernel** a la que se hace referencia en las últimas dos funciones, es la matriz filtro utilizada para ambas operaciones. Es idéntica en ambos casos y tiene un tamaño de 11x11.

Luego, para evitar que el contraste entre el objeto detectado y el fondo no sea demasiado pronunciado, se aplica un desenfoque de tipo gaussiano, utilizando la siguiente función:

cv2. GaussianBlur(**Imagen**, **(**3,3), 0)

El último paso dentro del algoritmo de procesamiento de la imagen es la binarización de la máscara. Esta acción convierte todos los píxeles con una magnitud inferior a cierto umbral, que este caso es 60, en un cero lógico (es decir, negro), y todos aquellos píxeles iguales o superiores a dicho umbral en un uno lógico (que en el caso de imágenes, es equivalente a 255, es decir, blanco). Esto se consigue mediante la siguiente función:

cv2.threshold(**Imagen**, 60, 255, cv2.THRESH\_BINARY)

Una vez la etapa de procesamiento ha terminado, se tiene una máscara que consiste en una imagen con un fondo completamente negro, donde los objetos color piel se encuentran representados por siluetas completamente blancas.

Utilizando esta nueva imagen, se aplica una función que se encarga de identificar los contornos de todos los objetos que se encuentren. Puesto que el número de objetos puede llegar a ser muy grande, se ordenan de mayor a menor por tamaño del área y se seleccionan los más grandes que tengan un área mínima de 3.000 píxeles. El número exacto de contornos a ser seleccionados es elegible por el usuario al comienzo de la ejecución del algoritmo. El conjunto de funciones utilizadas es el siguiente:

**Contornos**=cv2.findContours(**Imagen**,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

**Areas** = [cv2.contourArea(**c**) **for** **c** **in** **Contornos**]

**for** **i** **in** **range**(0, **Max\_Objetos**):

**Areas** = [cv2.contourArea(**c**) **for** **c** **in** **Contornos**]

**max\_index** = np.argmax(**Areas**)

**idx** = **Contornos** [**max\_index**]

**x**, **y**, **w**, **h** = cv2.boundingRect(**idx**)

**bboxs**.append((**x**, **y**, **w**, **h**))

La ejecución de este segmento de código extrae las coordenadas de cada uno de los rectángulos que encierra completamente el número máximo de contornos seleccionado, y los guarda en una matriz. Las coordenadas utilizadas para generar este rectángulo son: posición en el eje vertical y horizontal de la esquina superior izquierda del mismo, el alto y el largo. Este rectángulo se imprimirá posteriormente sobre la imagen original para identificar el objeto de interés en vivo.



Figura 23: Secuencia de imágenes mostrando procesamiento al identificar piel humana

Una vez conocidas las coordenadas de los objetos de interés, se procede a hacer su seguimiento para cada cuadrado subsiguiente. En este caso, al existir la posibilidad de tener que analizar más de un objeto a la vez, se utilizó la función “*Multitracker*”, y el *tracker* utilizado es del tipo MIL.

Para inicializar el *tracker* por primera vez (es decir, cuando el cuadro anterior no tenía ningún objeto detectado), se utiliza la siguiente función:

**for** **i** **in** **range**(0, **Max\_Objetos**):

**new\_tracker** = cv2.TrackerMIL\_create()

\_ = **trackers**.add(**new\_tracker**, **Imagen**, **bboxs**[**i**])

Este segmento de código genera una nueva variable **new\_tracker** de tipo MIL por cada nuevo objeto detectado, y lo agrega al arreglo de variables **trackers** junto con la imagen original y las coordenadas del objeto a seguir.

Una vez que el *tracker* está configurado, solo es necesario actualizarlo en cada nuevo cuadro de vídeo, y si se pierde de vista el objeto, el *tracker* se elimina. Para actualizarlo, se debe utilizar la función mostrada a continuación:

**tracking**, **bboxs** = **trackers**.update(**Imagen**)

La variable **tracking** es un booleano que indica si los objetos fueron encontrados exitosamente o no, mientras que en la variable **bboxs** se actualizan las nuevas coordenadas de los objetos. Adicionalmente, mientras el tracker esté activo, se imprime en la imagen original un rectángulo azul en las coordenadas obtenidas, como se mencionó anteriormente.

Utilizando dichas coordenadas, se procede recortar el objeto de interés de la imagen original. Esta imagen secundaria, de tamaño menor que la original y que contiene principalmente el objeto de interés, será la utilizada para la clasificación. Sin embargo, es necesario primero definir si dicho objeto se trata de un rostro humano o no.

Para ello, se procedió a utilizar el clasificador cascada de patrones binarios locales (*LPE Cascade* en inglés), incluido en la librería OpenCV. Este clasificador identifica si el objeto es similar a un rostro humano visto frontalmente. En caso negativo, se asume que se trata de una mano.

Para el caso del clasificador de expresiones faciales, la etapa de clasificación solo se lleva a cabo en caso de que se haya detectado un rostro, y en caso negativo, se procede a evaluar el siguiente objeto de interés. Por el contrario, en el caso del clasificador de señas del lenguaje de sordomudos, únicamente se procede a la etapa de clasificación cuando no se detecta ningún rostro en la imagen secundaria, mientras que, en caso de detectarse un rostro, se pasa al siguiente objeto de interés. Una vez se hayan analizado todos los objetos de interés en el cuadro de vídeo, y si no se ha detectado ningún objeto clasificable en este, se pasa a procesar el siguiente.

En ambos casos, la región de interés identificada se recorta de la imagen total, y se procesa exactamente de la misma manera que el dataset de entrenamiento de la red neuronal para la clasificación de expresiones faciales. Es decir, se convierte a escala de grises de 256 bits, se le aplica la función de contraste y se reduce su resolución. En el caso del clasificador de expresiones faciales, la reducción se hace a 32x32 píxeles, mientras que, para el clasificador de señas del lenguaje de sordomudos, se hace a 28x28 píxeles.

Una vez que la imagen ya tiene las mismas características de las imágenes del dataset de entrenamiento, se procede a utilizar la red neuronal correspondiente entrenada previamente.

El resultado de dicha clasificación va a ser un vector del mismo tamaño que el vector de etiquetas del dataset utilizado durante el entrenamiento. Los valores de cada uno de los números dentro del vector son directamente proporcionales a la magnitud en la que clasificador considera que la muestra pertenezca a cada clase en particular. Por lo tanto, el valor máximo en dicho vector corresponderá con la clasificación que la red neuronal ha realizado.

Por último, se utiliza este valor para expresar el resultado real (una expresión facial o una letra, según sea el caso) y se imprime en la imagen original para su visualización en vivo, y se pasa a la evaluación del siguiente cuadro de vídeo.

Todo el algoritmo explicado en esta sección se ejecuta de forma recursiva hasta que el usuario pulse la tecla ESC.

# RESULTADOS

Como se mencionó en el capítulo anterior, ambos datasets están divididos en una proporción 80% - 20%, donde el grupo mayor se utilizó para el entrenamiento de las redes neuronales artificiales, y el restante 20% se utilizó para validar la correspondiente red.

### Pruebas post-entrenamiento

Inmediatamente después del entrenamiento cada una de las redes neuronales, se procedió a probar su desempeño sobre el dataset de pruebas. Para ello, se procedió a clasificar todas las imágenes de cada dataset de prueba, y luego se comparó la clasificación obtenida con la etiqueta de cada una de ellas.

Durante esta etapa del proyecto se probaron diferentes configuraciones para las redes neuronales, y los parámetros en los puntos 4.4.2 y 4.4.3 se corresponden con la configuración que mostró los mejores resultados, para cada uno de los casos.

Dichos resultados, para cada una de las redes, se muestran a continuación:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Porcentaje de acierto del clasificador de expresiones faciales | | | | | | | |
| Neutral | Alegría | Tristeza | Ira | Disgusto | Miedo | Sorpresa | **Total** |
| 77% | 77% | 80% | 76% | 80% | 79% | 74% | **79%** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Porcentaje de acierto del interpretador del lenguaje de sordomudos | | | | | | | | | | | | |
| A | B | C | D | E | F | G | H | I | K | L | M | N |
| 99% | 95% | 97% | 96% | 95% | 99% | 93% | 91% | 78% | 90% | 95% | 79% | 70% |
| O | P | Q | R | S | T | U | V | W | X | Y | **Total** | |
| 77% | 98% | 90% | 69% | 79% | 58% | 69% | 82% | 90% | 91% | 79% | **87%** | |

Figura 24: Tablas de resultados de pruebas después del entrenamiento

### Pruebas en vivo

Una vez que se seleccionaron las redes con el mejor desempeño, las mismas se aplicaron respectivamente en cada uno de los programas que utilizan el algoritmo explicado en el punto 4.5, con el objetivo de que realizaran la clasificación de expresiones faciales y señas, dependiendo del caso.

#### Clasificación de expresiones faciales humanas

Al comenzar a ejecutar el programa totalmente integrado, se le pedirá al usuario que indique el número máximo de rostros que se analizarán por cada cuadro de vídeo. Se debe tomar en cuenta que entre mayor sea este número, el procesamiento de cada cuadro de vídeo será más exigente y tomará más tiempo.

A continuación, para cada rostro identificado en cada cuadro de vídeo, se aplicará el clasificador de expresiones faciales. El desempeño general del mismo depende de varias variables, entre las que destacan la edad del sujeto, su género, etnia, iluminación del ambiente, distancia hasta la cámara, ángulo del rostro, calidad del vídeo, entre otras.

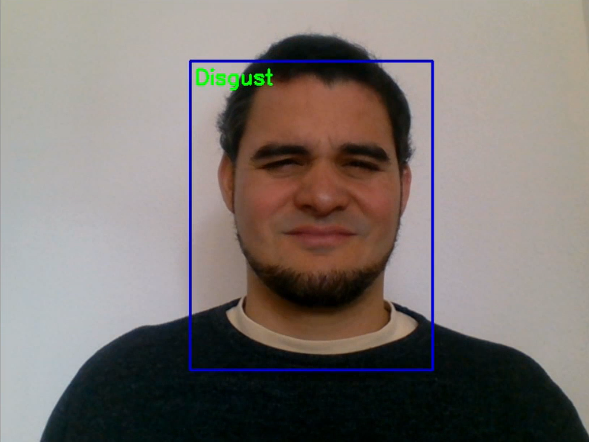


Figura 25: Sujeto uno mostrando expresiones de “ira” y “disgusto” (de izquierda a derecha)

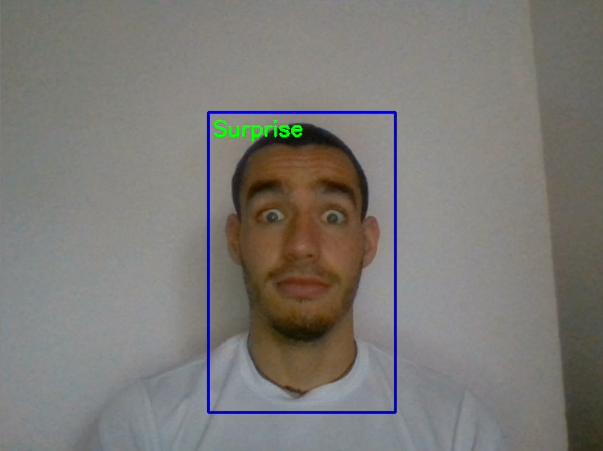


Figura 26: Sujeto dos mostrando expresiones de “tristeza” y “sorpresa” (de izquierda a derecha)

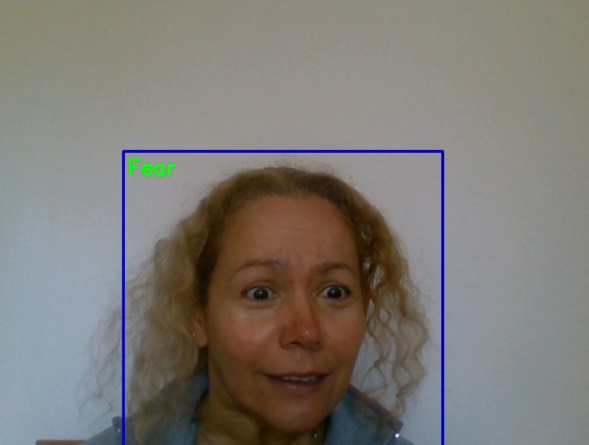


Figura 27: Sujeto tres mostrando expresiones de “alegría” y “miedo” (de izquierda a derecha)

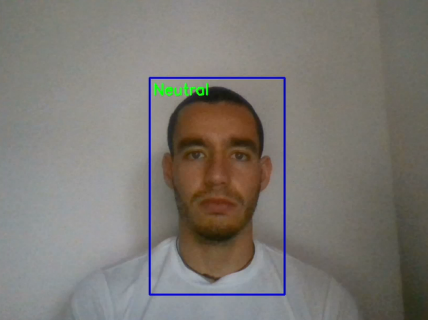
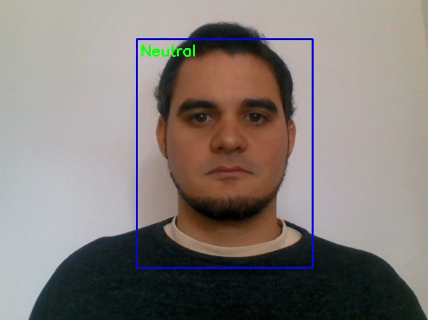


Figura 28: Los tres sujetos de prueba mostrando la expresión “neutral”

Al experimentar con distintos sujetos de estudio, se pudieron apreciar diferencias en el margen de acierto de algunas expresiones faciales sobre otras, dependiendo de la edad y género del sujeto, así como de la calidad de la iluminación. Esto se debió principalmente al hecho de que en el dataset de entrenamiento predominaban sujetos de entre 20 y 30 años, mayoritariamente caucásicos y con una predominancia femenina (ver punto 4.2.1).

#### Clasificación de señas del lenguaje de sordomudo americano

Al realizar la ejecución de este programa, primero se le pedirá usuario que indique el número máximo de manos que serán analizadas por cada cuadro de vídeo. Es indistinto el uso de la mano derecha o izquierda durante este experimento, pero se debe tomar en cuenta que entre mayor sea el número de manos seleccionada, el procesamiento de cada cuadro de vídeo será más exigente.

Luego, para cada mano identificada en cada cuadro de vídeo, se aplicará el clasificador de señales. El desempeño de dicho clasificador puede variar dependiendo principalmente de la iluminación ambiente, el ángulo de la mano, distancia hasta la cámara, la calidad del vídeo y qué tan diferente es la señal en cuestión con respecto a otra en el alfabeto. Además, al utilizarse el color de la piel para identificar la extremidad, es obligatorio el uso de indumentarias de un color diferente al de la piel y con mangas largas para el correcto funcionamiento del algoritmo.

A continuación, se muestran algunos ejemplos de detección y clasificación para ocho letras diferentes:

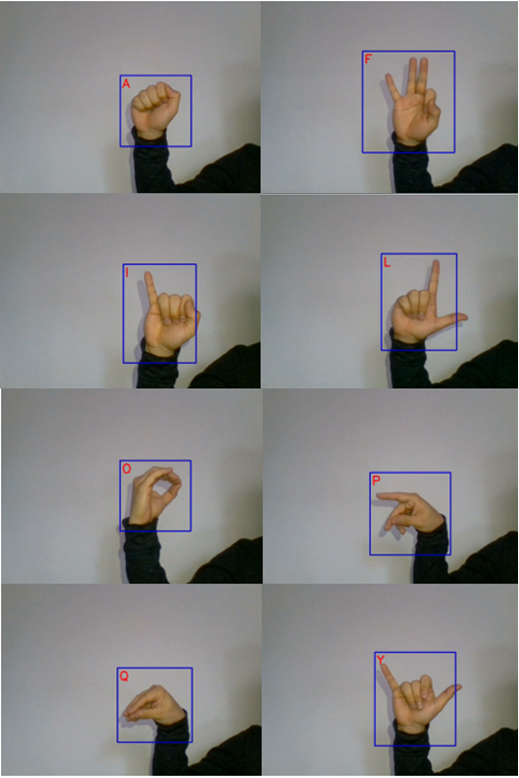


Figura 29: Sujeto mostrando las señas para las letras “A”, “F”, “I”, “L”, “O”, “P”, “Q”, “Y” (de izquierda a derecha y de arriba abajo)

Al experimentar con distintos sujetos y señas, se puede observar una marcada diferencia en la capacidad de acierto entre diferentes señas. Los principales casos de desacierto se deben a la similitud entre dos o más señas. Entre ellas destacan las señas para las letras “A”, “M”, “N” y “S”; “F”, “V” y “W”; “P” y “Q”; “D”, “K” y “R”. La figura 3 permite realizar esta comparación.

De igual manera, el clasificador presentó la misma precisión a la hora de evaluar señas realizadas con la mano derecha o la mano izquierda, con lo que la etapa de expansión de datos demostró ser útil.

# POSIBLES APLICACIONES

La intención de este proyecto es dotar a los robots con nuevas herramientas que les permitan una mejor comunicación con seres humanos.

Durante el desarrollo del mismo, se mantuvo como principal objetivo a futuro su aplicación como un servicio en la nube al que pudieran acceder robots para la interpretación de las imágenes que estuviese recopilando en vivo durante su interacción con seres humanos. De esta manera, los robots que accediesen a dicho servicio serían capaces de mejorar el entendimiento de ciertas expresiones lingüísticas ambiguas, pero cuyo significado es fácilmente interpretable dependiendo de la expresión facial que muestra el interlocutor. De igual forma, dichos robots también serían capaces de comunicarse visualmente con cualquier persona que utilice el lenguaje de sordomudos americano.

El hecho de que ambos programas se hayan desarrollado utilizando el lenguaje Python, permite que su uso no esté limitado a los modelos Pepper y Nao que se tuvieron en cuenta durante su desarrollo, sino con la gran mayoría de robots que se están comercializando en la actualidad.

Adicionalmente, en el caso del interpretador del lenguaje de señas de sordomudo americano, sería posible desarrollar una aplicación móvil utilizando su algoritmo como motor, y que permita a otros seres humanos ser capaces de comunicarse más efectivamente con personas que utilicen este lenguaje, en caso de no conocerlo.

# CONCLUSIONES Y REFLEXIONES FINALES

La rama de la inteligencia artificial sigue avanzando a gran velocidad y mezclándose con distintas tecnologías, permitiendo un crecimiento sostenido en su número de aplicaciones que en el día a día son utilizadas por casi todas las personas. Una de las tecnologías que sigue esta línea, y que fue la principal motivación de este trabajo, es la robótica, especialmente aquella enfocada a interacciones sociales con seres humanos.

Los robots humanoides que se encuentran en el wavespace de Ernst & Young en Madrid tienen contacto directo diariamente con los empleados de la organización, clientes y demás visitantes del Centro de Excelencia. Todas las distintas habilidades de estos robots fueron desarrolladas internamente en la firma, y les han permitido ir rompiendo la brecha entre el robot visto como una herramienta de trabajo o un juguete, para convertirlos en unos de los anfitriones más apreciados y buscados dentro y fuera de la empresa. Ada y Alan (como se les conocen) son asiduos participantes en las distintas exposiciones de EY tanto dentro como fuera de España, así como en convenciones y foros, tratando de demostrar de qué son capaces.

Este Trabajo Final de Máster está permitiendo ampliar el abanico de capacidades de estos robots, al introducir la posibilidad de utilizar uno de los programas desarrollados para interpretar directamente las expresiones faciales que los seres humanos utilicen al comunicarse con ellos, además de abrir las puertas para el desarrollo de una aplicación que utilice el clasificador del otro programa como motor que permita la comunicación con personas con déficit de audición.

Sobre el programa para la clasificación en vivo de expresiones faciales, se puede decir que se lograron obtener resultados por encima de los esperados, al mostrar un porcentaje de acierto sobre la prueba de validación del 79%. Este número se considera alto al tomar en cuenta que el tamaño de la base de datos utilizada para el entrenamiento es relativamente pequeña para los estándares del *Machine Learning*. De igual manera, dicha base de datos muestra una homogeneidad por debajo de la deseada, al tener una marcada mayor representación de la etnia caucásica sobre todas las demás, así como una mayor proporción de sujetos de sexo femenino y una fuerte tendencia a edades comprendidas entre los 18 y 30 años. Esto se debe principalmente a que la mayoría de los voluntarios eran estudiantes del Departamento de Psicología de la Universidad de Columbia, en EE.UU.

Otro aspecto para tomar en cuenta es que se observó una predominancia en la correcta interpretación de las expresiones correspondientes a “Alegría” y “Tristeza” por encima de todas las demás durante la clasificación en vivo. A pesar de no poseer evidencia concreta de la causa de esto, es posible interpretar que podría deberse al hecho de que son las expresiones faciales que los seres humanos están más acostumbrados a realizar tanto de forma natural, como actuada. Esta situación podría producir que los sujetos de prueba sean más experimentados al mostrarlas frente a la cámara. Otra causa más concreta que podría tener incidencia en dicha afirmación, y que puede estar ligada a lo mencionado anteriormente, es el hecho de que las expresiones faciales asociadas a estas emociones son más homogéneas entre sí respecto a las otras expresiones. Esta homogeneidad es clave a la hora de identificar características y patrones durante el entrenamiento de la red neuronal, e incidir directamente en su capacidad de acierto.

Por otra parte, el programa para la interpretación en vivo de señas del lenguaje de sordomudos, en su versión americana, presentó unos resultados en la prueba de validación muy por encima de lo que se esperaba, con un porcentaje medio de aciertos del 87%, y con hasta 8 letras que superaron el 95% de aciertos. La principal diferencia entre los resultados de este clasificador y el anterior radican principalmente en la calidad de la base de datos utilizada durante el entrenamiento: en este caso, las muestras se encontraban todas ya preprocesadas para ser utilizadas directamente en el entrenamiento, y su número sí se adecúa a los estándares del *Machine Learning*, siendo más de 50 veces más amplia.

Sin embargo, es importante mencionar que algunas letras presentaron un porcentaje de aciertos por debajo de lo esperado. El patrón común entre estas letras es que la seña establecida para ellas es muy parecida a otras señas, produciendo cierta ambigüedad al compartir muchas características, por lo que la resolución de 28x28 píxeles no es suficiente para ser interpretadas, en algunos casos ni siquiera por un ser humano.

Tomando en cuenta lo mencionado anteriormente, se pudo observar durante la clasificación en vivo que el rendimiento es superior al del clasificador de expresiones faciales, si no se toman en cuenta las señas ambiguas. Esta situación podría cambiar si se generara una nueva base de datos para entrenar al clasificador de expresiones fáciles, que cuente con un tamaño y homogeneidad similar a la del clasificador de señas.

Además, trabajos futuros podrían permitir el uso de del clasificador de señas para la elaboración de un programa más complejo que sea capaz de ir capturando las señas que se van interpretando y formar palabras y frases en vivo, de forma tal que esta tecnología pueda ser utilizada no sólo por robots, sino también por otros seres humanos.

Finalmente, se puede decir que los objetivos planteados para este trabajo fueron conseguidos. Ambos clasificadores muestran un rendimiento medio que supera lo esperado inicialmente, y sus funcionalidades son aplicables a los modelos de robots dentro del wavespace de Ernst & Young en Madrid. Se espera que un futuro próximo, la introducción de ambos programas entre los servicios en la nube que alimentan a los robots del wavespace sea realizada, de forma tal que puedan ser dotados de estas nuevas capacidades.

# MEJORAS Y FUTUROS DESARROLLOS

A pesar de que los resultados obtenidos fueron satisfactorios, una vez finalizado el proyecto se han identificado varios aspectos donde se pueden aplicar mejoras de cara a próximos desarrollos en esta área.

Entre ellos se destacan los siguientes:

* Utilizar un dataset más grande y diverso para el entrenamiento de la red neuronal artificial que identifica expresiones faciales.
* Recopilar dentro del dataset mencionado anteriormente imágenes de rostros con expresiones faciales más naturales y, por lo tanto, menos exageradas.
* Incluir en la ejecución de ambos programas una modalidad que guarde ciertos cuadros para que el usuario evalúe si el clasificador acertó o no, y en base a ello reentrenarlo.
* Desarrollar un método que además de interpretar las señas del lenguaje de sordomudo, vaya generando palabras a partir de las letras identificadas.
* Incluir en las pruebas a sujetos que utilicen el lenguaje de señas de forma regular, de forma tal que se puedan evaluar casos más naturales.
* Desarrollar e incluir un clasificador de manos, de forma tal que no se tenga que asumir que un objeto es una mano sólo porque no fue identificado previamente como un rostro.
* Evaluar un rediseño general de los algoritmos para buscar reducir el procesamiento requerido durante las ejecuciones y mejorar la calidad final del mismo.

Se espera que algunas de las mejoras señaladas puedan ser desarrolladas antes de la aplicación de este proyecto a robots comerciales en los próximos meses.

# BIBLIOGRAFÍA

|  |  |
| --- | --- |
| [1] | C. Fresneda, «El Mundo,» 10 Junio 2014. [En línea]. Available: https://www.elmundo.es/ciencia/2014/06/09/539589ee268e3e096c8b4584.html. [Último acceso: Febrero 2019]. |
| [2] | «Aprendizaje automático,» [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje\_autom%C3%A1tico. [Último acceso: Marzo 2019]. |
| [3] | «Artificial Neural Networks as Models of Neural Information Processing: Frontiers Research Topic,» [En línea]. Available: https://www.frontiersin.org/research-topics/4817/artificial-neural-networks-as-models-of-neural-information-processing. [Último acceso: Febrero 2018]. |
| [4] | «Perceptrón Multicapa,» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/Perceptr%C3%B3n\_multicapa. [Último acceso: Marzo 2019]. |
| [5] | T. Kohonen, «Self-Organized Formation of Topologically Correct Feature Maps,» de *Biological Cybernetics*, 1982, p. 43. |
| [6] | «Redes Neuronales Convolucionales,» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/Redes\_neuronales\_convolucionales. [Último acceso: Marzo 2019]. |
| [7] | D. H. Hubel y T. N. Wiesel, «Receptive fields of single neurons in the cats striate cortex,» de *The Journal of Physiology*, 1959, p. 148. |
| [8] | E. Gazar, "Ehsangazar.com," 6 Agosto 2018. [Online]. Available: https://ehsangazar.com/object-tracking-with-opencv-fd18ccdd7369. [Accessed Noviembre 2018]. |
| [9] | W. Berger, "Willberger.org," 2018. [Online]. Available: http://www.willberger.org/cascade-haar-explained/. [Accessed Marzo 2019]. |
| [10] | «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/An%C3%A1lisis\_de\_componentes\_principales. [Último acceso: Marzo 2019]. |
| [11] | «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Python. [Último acceso: Marzo 2019]. |
| [12] | A. Rosebrock, «Detecting Skin in Images & Video Using Python and OpenCV,» Pyimagesearch.com, 2016. [En línea]. Available: https://www.pyimagesearch.com/2014/08/18/skin-detection-step-step-example-using-python-opencv/. [Último acceso: Noviembre 2018]. |
| [13] | J. Chen, «Sign Language Recognition with Unsupervised Feature Learning,» 2014. [En línea]. Available: http://cs229.stanford.edu/proj2011/ChenSenguptaSundaram-SignLanguageGestureRecognitionWithUnsupervisedFeatureLearning.pdf/. [Último acceso: Octubre 2018]. |
| [14] | «Image Preprocessing,» Keras, [En línea]. Available: https://keras.io/preprocessing/image/. [Último acceso: September 2019]. |

# ANEXOS

Todo el código final desarrollado durante este proyecto se encuentra publicado abiertamente en GitHub. Dentro de cada una de las carpetas, se puede encontrar el código que es necesario ejecutar para ver el algoritmo en funcionamiento, junto con una carpeta (CNN) donde se encuentra la red neuronal utilizada para la clasificación, y otra carpeta (Training) donde se encuentra el código utilizado para el entrenamiento de las redes neuronales artificiales:

* Detector de expresiones faciales:

<https://github.com/pzampella/Expression-and-Sign-Detector/tree/master/Emotion_Analytics>

* Interpretador de señas del lenguaje de sordomudos:

<https://github.com/pzampella/Expression-and-Sign-Detector/tree/master/Sign_Detection>