Escuela Politécnica Superior 

Trabajo Fin de Grado

Grado En ingeniería electrónica industrial

Dispositivo empotrado para traducción de lenguaje de signos usando técnicas ML/DL

Autor: Santiago Moreno Ordoñez.

Tutores: Luis Muñoz Saavedra, Francisco Luna Perejón.

Julio 2023

Trabajo Fin de Grado

Ingeniería Electrónica Industrial

Dispositivo empotrado para traducción de lenguaje de signos usando técnicas ML/DL

Autor:

Santiago Moreno Ordoñez

Tutores:

Luis Muñoz Saavedra

Francisco Luna Perejón

Dpto. de Arquitectura y Tecnología de Computadores

Escuela Politécnica Superior

Universidad de Sevilla

Sevilla, 2023

AGRADECIMIENTOS

Se quieren dar agradecimientos a todas las personas que me han acompañado en este trayecto, a mis compañeros de clase por ayudarme tanto en mi crecimiento académico como personal, en especial a las personas que han sido cercanas y que hemos acabado superando esta etapa. Además, he de agradecer a mi tutor del TFG que me ha ayudado siempre que lo he necesitado y ha sido muy atento preocupándose por hacer un seguimiento.

También quiero dar gracias a mi madre por ser la persona que más ha confiado y quien más me ha apoyado.

Para finalizar, se hace una mención de agradecimiento al FabLab de la Universidad de Sevilla por prestar la Raspberry Pi 4 y la cámara Logitech C920 HD Pro.

RESUMEN

En este trabajo de fin de grado se va a explicar el desarrollo de un sistema de reconocimiento del lenguaje de signos utilizando técnicas de Machine Learning (ML), Deep Learning(DL) y visión por computadora. El objetivo principal de este proyecto es diseñar e implementar un programa que permita capturar y clasificar imágenes, en este caso, que contengan los gestos de las manos que corresponden a el abecedario del lenguaje de signos inglés. Para ello, se emplea una red neuronal convolucional para el entrenamiento y la clasificación de las imágenes capturadas con una cámara.

Además, se hará uso de una Raspberry Pi junto con una cámara y un display para ejecutar el programa y mostrar los resultados en un entorno visual. Los resultados obtenidos demuestran una alta precisión en el reconocimiento de las letras del lenguaje de signos, lo que valida la eficacia del sistema propuesto.

PALABRAS CLAVE

CNN = Ren Neuronal Convolucional.

AN = Neurona artificial.

Dataset = Conjunto de datos que se van a usar en el entrenamiento del modelo.

Display = Dispositivo que muestra información de forma visual.

ML = Aprendizaje automático.

DL = Aprendizaje profundo.

Dropout = Tasa de abandono.

ARM = Arquitectura de máquina RISC avanzada.

ROI = Área de interés.

ABSTRACT

In this undergraduate thesis, the development of a sign language recognition system using Machine Learning (ML), Deep Learning (DL), and computer vision techniques will be explained. The main objective of this project is to design and implement a program that allows for capturing and classifying images containing hand gestures corresponding to the English sign language alphabet. For this purpose, a convolutional neural network is employed for training and classifying the captured images using a camera.

Furthermore, a Raspberry Pi, along with a camera and a display, is utilized to run the program and display the results in a visual environment. The obtained results demonstrate high accuracy in recognizing sign language letters, thus validating the effectiveness of the proposed system.

KEYWORDS

CNN = Convolutional Neural Network

AN = Artificial Neuron

Dataset = Set of data used for model training

Display = Device that shows information visually

ML = Machine Learning

DL = Deep Learning

Dropout = Dropout rate

ARM = Advanced RISC Machine

ROI = Region of Interest

ÍNDICE

1. INTRODUCCIÓN

1.1. Contexto y motivación

El lenguaje de signos es una forma de comunicación para las personas con discapacidad auditiva y/o del habla, ya que les permite expresarse y establecer una comunicación con los demás. Sin embargo, hay muchas barreras en la comunicación entre las personas que usan el lenguaje de signos y las que no lo comprenden.

En este contexto, el desarrollo de un sistema de reconocimiento del lenguaje de signos adquiere importancia. Este proyecto pretende facilitar la comunicación entre las personas que usan dicho lenguaje y el resto de la sociedad, permitiendo una interacción más fluida y sin barreras.

La motivación principal es utilizar tecnologías de Machine Learning y Deep Learning para crear un sistema que pueda identificar y reconocer los gestos de las manos correspondiente al abecedario del lenguaje de signos. Al contar con un sistema los más preciso y confiable posible, las personas con la discapacidad mencionada podrían tener una vía más para comunicarse de forma más sencilla, ya sea en entornos como escuelas, hospitales, espacios públicos y en sus relaciones personales.

Además, el uso de la Raspberry Pi como plataforma de implementación ofrece una solución portátil y de bajo costo, lo que permite que el sistema pueda ser accesible y usado en diferentes situaciones.

1.2. Objetivos

El objetivo general es crear un sistema que usando una placa Raspberry Pi, una cámara, un display y un programa escrito en Python, pueda clasificar las imágenes que se capturan y al pasarlas por un clasificador reconozca qué letra del lenguaje de signos contiene para poder imprimir en el display dicha letra, para ello se establecen los siguientes objetivos específicos:

- Investigar y analizar el estado de las técnicas de reconocimiento del lenguaje de signos y visión por computadora.
- Diseñar e implementar un programa que permita la captura y clasificación de imágenes, las cuales contiene un gesto representando el SL.
- Realizar la adquisición y preparación de un dataset adecuado para el entrenamiento y evaluación del sistema de clasificación de imágenes.
- Entrenar una red neuronal convolucional (CNN) utilizando el dataset preparado para el reconocimiento de los gestos.
- Evaluar el rendimiento del sistema de clasificación mediante funciones y gráficos.
- Implementar el sistema en una placa Raspberry Pi 4 junto con una cámara y un display, para uso en un entorno práctico.
- Validar el sistema a través de pruebas, capturando imágenes de un usuario realizando un gesto y verificando la precisión.
- Realizar un análisis de los resultados obtenidos y discutir las conclusiones, limitaciones y posibles mejoras del sistema.

1.3. Estructura de la memoria

La memoria del presente Trabajo de Fin de Grado se organiza en los siguientes capítulos:

- **Capítulo 2:** Se presenta los fundamentos teóricos necesarios para comprender el contexto del proyecto. Se abordan conceptos como el lenguaje de signos y su importancia en la comunicación para las personas con discapacidad auditiva y/o del habla, así como las técnicas de Machine Learning (ML) y Deep Learning (DL) utilizadas en el desarrollo del sistema.
- **Capítulo 3:** Se desarrolla en detalle la realización del TFG, comenzando por el diseño del sistema. Se explican los programas desarrollados, su estructura y funcionamiento, así como la arquitectura del sistema tanto en términos de hardware como de software.

- **Capítulo 4:** Se presentan los resultados y la evaluación del proyecto. Se analizan los resultados obtenidos en la etapa de entrenamiento del modelo y se evalúa su rendimiento en términos de precisión y otras métricas relevantes. Se incluyen gráficos y tablas que ilustran los resultados de las pruebas realizadas.
- **Capítulo 5:** Se exponen las conclusiones derivadas del trabajo realizado. Se resumen los logros alcanzados en el TFG, se reflexiona sobre el proceso de desarrollo y se distinguen las limitaciones encontradas durante el proyecto.
- **Capítulo 6:** Se enfoca el proyecto en el futuro y las posibles mejoras o ampliaciones que se podrían llevar a cabo.
- **Finalmente:** Se incluyen las referencias bibliográficas utilizadas en la investigación y el desarrollo del TFG, así como los anexos que contienen el código fuente de los programas, capturas de pantalla y fotografías del sistema en funcionamiento, y otros materiales relevantes para la comprensión del proyecto.

2. FUNDAMENTOS TEÓRICOS

2.1. Lenguaje de signos y su importancia

El lenguaje de signos es un sistema de comunicación que usa el canal visual-gestual (a diferencia de las lenguas habladas que usan el oral-auditivo) usado por las personas sordas y con discapacidad auditiva para expresar y recibir información. Este sistema se basa en el uso de gestos, movimientos de manos, expresiones faciales y posturas corporales para transmitir significados.

La lengua de signos desempeña un papel importante en la vida de las personas sordas, ya que ofrece una alternativa de comunicación con el resto de las personas y promueve la inclusión social. Además, ayuda al desarrollo lingüístico y cognitivo, mejorando las capacidades sociales e incluso ayudando a un correcto desarrollo del pensamiento crítico. Otro factor para tener en cuenta es que da acceso a que las personas sordas tengan acceso a la información, servicios y derechos básicos en igualdad de condiciones, como la comunicación en entornos educativos, médicos, legales y administrativos, entre otros.

Cabe destacar que el lenguaje de signos tiene relación con su situación geográfica puesto que puede variar dependiendo del sitio donde se hable, por ejemplo, en las zonas de habla hispana se hace uso de la letra “ñ”, sin embargo, en otras zonas está incluida en su alfabeto.

2.2. Aprendizaje automático

El aprendizaje automático, de forma general, implica el uso de algoritmos informáticos para encontrar la estructura en los datos. Este estudio se basa en el llamado “aprendizaje supervisado” el cual resuelve problemas conocidos y utiliza un conjunto de datos etiquetados para entrenar un algoritmo, en otras palabras, se aprende un mapeo de los datos de entrada a una etiqueta de salida. Aquí, la estructura de los datos se representa como un conjunto de características extraídas de los datos de entrada. Para este Trabajo de Fin de grado, los datos de entrada son imágenes de manos los gestos correspondientes a el alfabeto del lenguaje de

signos, y el resultado es una etiqueta de salida donde se ve la predicción hace el modelo identificando a qué letra corresponde el gesto realizado.

Uno de los principales aspectos en el aprendizaje automático es que el modelo, una vez entrenado, pueda predecir no solo con el conjunto de datos con el que aprendió, sino también en ejemplos no incluidos en estos datos. A esto se le suele llamar "generalizabilidad" de un modelo. Para ello tenemos 2 subconjuntos de datos:

- uno llamado "datos de entrenamiento", que se destina al entrenamiento del modelo,
- otro llamado "datos de prueba", que son distintos a los de entrenamiento y se encargan de comprobar que el modelo funcione con datos nuevos.

Cada modelo de aprendizaje automático tiene un conjunto de propiedades características las cuales se pueden modificar antes del proceso de entrenamiento, a estas se les llama "hiperparámetros", el rendimiento del modelo puede variar mucho según la configuración de estas propiedades por lo que hay que intentar conseguir la configuración óptima.

2.3. Identificación de imágenes

Todo ser vivo tiene el instinto de identificar y clasificar las cosas que le rodean según sus sentidos. En el caso de este trabajo de visión artificial, esta clasificación pretende asignar una categoría a una o varias etiquetas reconociendo las características dominantes de la imagen. La clasificación es el proceso por el cual se asigna una clase o categoría predefinida en según sus características observables.

La clasificación de imágenes se puede realizar de varias formas, por ejemplo: Árboles de decisiones, redes neuronales, entre otros; A continuación, se representa un diagrama general de identificación de imágenes.

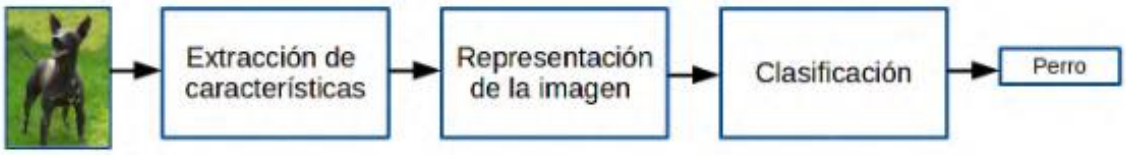


Figura 2.3.1 Diagrama general de identificación de imágenes.

- **Extracción de características:** Entendiendo como características a los elementos que describen un objeto, como puede ser su tamaño, forma, color, etc. Se deben extraer los patrones o características de la mejor forma posible para que permitan identificar la imagen para llegar a tener un buen rendimiento en la clasificación. Estas características se suelen encontrar en áreas con cambios de contraste o en los contornos de los objetos. Esta fase es importante ya que constituye el conocimiento previo del clasificador, y su mal funcionamiento puede hacer que el clasificador de imágenes no funcione apropiadamente.

- **Representación de la imagen:** Para poder tratar una imagen, es común representarla como un arreglo numérico de forma que sumando o combinando los vectores se describen las áreas de interés. También hay que recalcar que para esta representación en muchos casos es recomendable normalizar los datos para un mejor rendimiento.

- **Clasificación:** En la fase final, se clasifica la imagen asignándole la categoría correspondiente en base al entrenamiento previo, el cual ha reunido las características comunes de los objetos de estudio, siendo capaz de reconocer las similitudes y realizar su clasificación.

Para la clasificación de imágenes existen diferentes clasificadores, a continuación, se van a describir brevemente algunos de ellos:

- **Clasificador paramétrico:** Estos clasificadores están basados en funciones de densidad de probabilidad, como podría ser una distribución normal.

- **Clasificador no paramétrico:** A diferencia de los anteriores, evitan el uso de parámetros estadísticos para realizar la categorización.

2.4. Redes convolucionales (CNN)

2.4.1. Algoritmos de aprendizaje

Antes de especificar las redes convolucionales, se va a explicar brevemente algunos de los algoritmos de aprendizaje que existen:

- **Máquinas de soporte vectorial:** Son algoritmos de aprendizaje supervisado que usan condiciones lineales para separar clases entre sí. Estas máquinas, también llamadas SVM (por sus siglas en inglés), resuelven problemas de clasificación y regresión.

- **Árboles de decisiones:** Se forma por "árboles" de decisiones que crean ramificaciones de forma jerárquica. De la misma forma intenta dividir los datos de entrenamiento para tener el mayor número de diferencias entre clases usando diferentes nodos. Su principal objetivo es el aprendizaje inductivo a partir de observaciones y construcciones lógicas, es uno de los más utilizados.

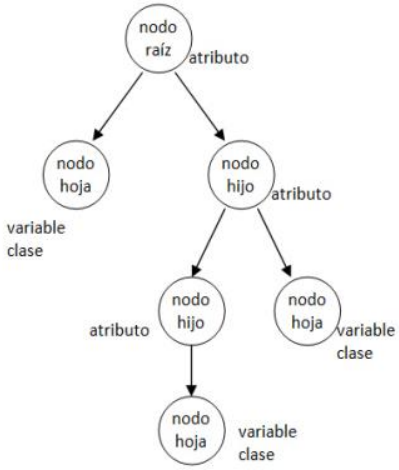


Figura 2.4.1.1 Estructura árbol de decisiones.

- **Algoritmos genéticos:** Son algoritmos de búsqueda y aprendizaje, utilizan el cruzamiento y la mutación para generar nuevos elementos. Su eficiencia depende de los parámetros establecidos como "poblaciones sucesivas".

- **K-means:** Es un algoritmo de agrupamiento en cual selecciona un número de objetos al azar y después trata de encontrar los elementos más cercanos haciendo cálculos de distancia.

Ahora, una vez introducidos algunos de los algoritmos que se pueden utilizar, se va a explicar más en profundidad las redes neuronales.

2.4.2. Neuronas biológicas

Los algoritmos de redes neuronales son modelos matemáticos inspirados en las neuronas biológicas, intentando asemejar el comportamiento al cerebro humano, por tanto, se va a introducir brevemente una neurona biológica. Su estructura, es un cuerpo celular (soma) que tiene el núcleo del que se extienden las dendritas (terminales receptoras) y el axón (elemento transmisor). La unión entre el axón y la dendrita de dos neuronas se llama sinapsis.

La comunicación entre neuronas está basada en terminales bio-químicas en el axón, que se activan liberando neurotransmisores que son captados por la dendrita.

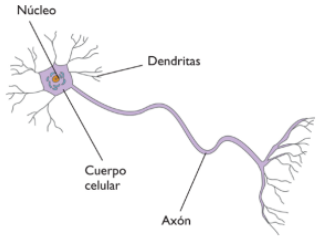


Figura 2.4.2.1 Estructura de una neurona

2.4.3. Neuronas artificiales

Como se ha mencionado antes, una neurona artificial intenta copiar la estructura de las redes neuronales biológicas, con el fin de conseguir una funcionalidad parecida. Hay 3 conceptos importantes que se van a emular:

- **Procesamiento paralelo:** neuronas operando en paralelo sobre la totalidad de la imagen.
- **Memoria distribuida:** Redundancia en el almacenamiento, para evitar pérdida de información en caso de que se dañe una sinapsis.
- **Adaptabilidad al entorno:** Para generalizar conceptos a partir de casos particulares.

En la práctica lo que se utiliza en Deep Learning son estas redes neuronas artificiales, dando la

posibilidad de clasificar por ejemplo imágenes o voz a nivel casi humano.

Una red neuronal sigue la estructura que se muestra en la imagen siguiente:

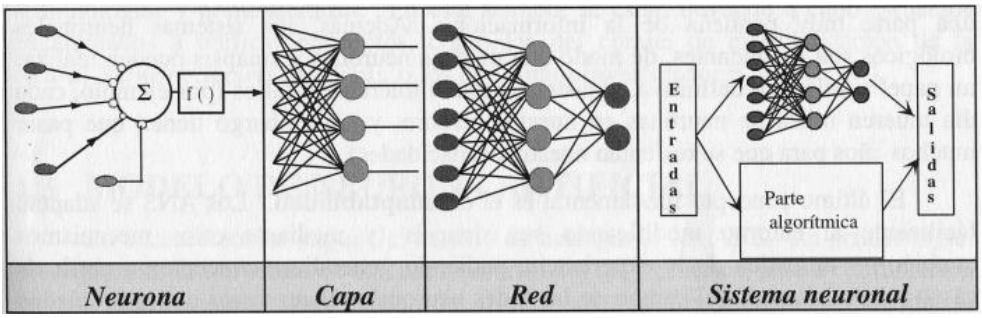


Figura 2.4.3.1 Estructura de una red neuronal artificial

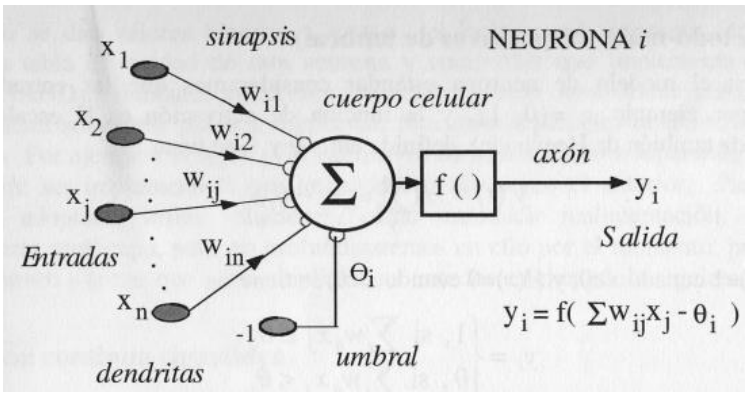


Figura 2.4.3.2 Estructura de una neurona

El elemento fundamental de una red es la neurona, también llamada “nodo” o “perceptrón”, está compuesta por componentes principales: entradas, pesos y una función de activación.

Las entradas son las señales o características que se proporcionan a la neurona (en el caso de este trabajo se introducen como un arreglo numérico). Las entradas tienen asociado un peso que se usa para la salida, dichos pesos se ajustan en el entrenamiento de la red para optimizar su rendimiento.

La función de activación se aplica a la suma ponderada de las entradas y los pesos introduciendo la no linealidad. Permite a la red aprender características o patrones complejos

en los datos. Esta función de activación representa simultáneamente la salida de la neurona y su estado de activación.

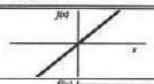
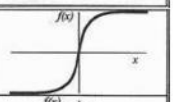
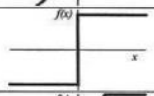
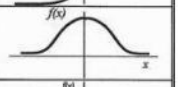
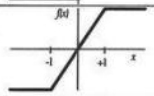
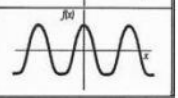
	Función	Rango	Gráfica		Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$		Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	[0, +1] [-1, +1]	
Escalón	$y = sign(x)$ $y = H(x)$	{-1, +1} {0, +1}		Gaussiana	$y = Ae^{-Bx^2}$	[0,+1]	
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \le x \le -l \\ +1, & si\ x > +l \end{cases}$	[-1, +1]		Sinusoidal	$y = A\,\mathrm{sen}(\omega x + \varphi)$	[-1,+1]	

Figura 2.4.3.3 Algunas de las funciones de activación más usadas

Otra de las funciones de activación más populares es la función ReLU (Rectified Linear Unit), su funcionamiento es tal que, si el valor de entrada es mayor que cero la salida es igual al valor de entrada; de contrario, la salida es cero. Es decir, activa la neurona sólo si la entrada es positiva y pone a cero los valores negativos.

Si se representa el modelo matemático como un sistema eléctrico discreto que lo implementa, se tiene:

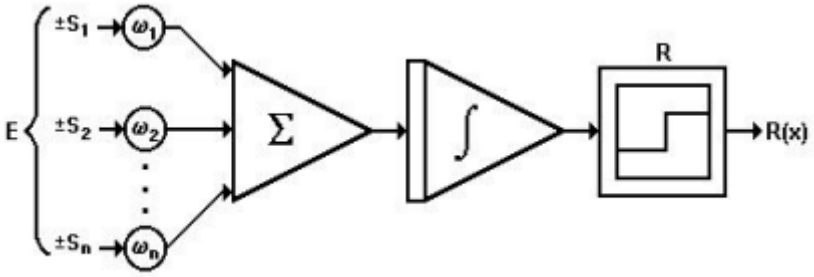


Figura 2.4.3.4 Modelo electrónico con componentes discretos.

2.4.4. Redes Neuronales Artificiales (ANN)

Las neuronas artificiales (AN) se pueden estructurar con capas que interconectan las neuronas por medio de las sinapsis como se puede ver en la figura 2.4.4. El comportamiento de la red se determina por la estructura de las conexiones sinápticas.

Las redes se pueden formar por una capa (monocapa) o más (multicapa), además, se pueden diferenciar según el sentido de la información, pudiendo ser unidireccional o con

realimentación.

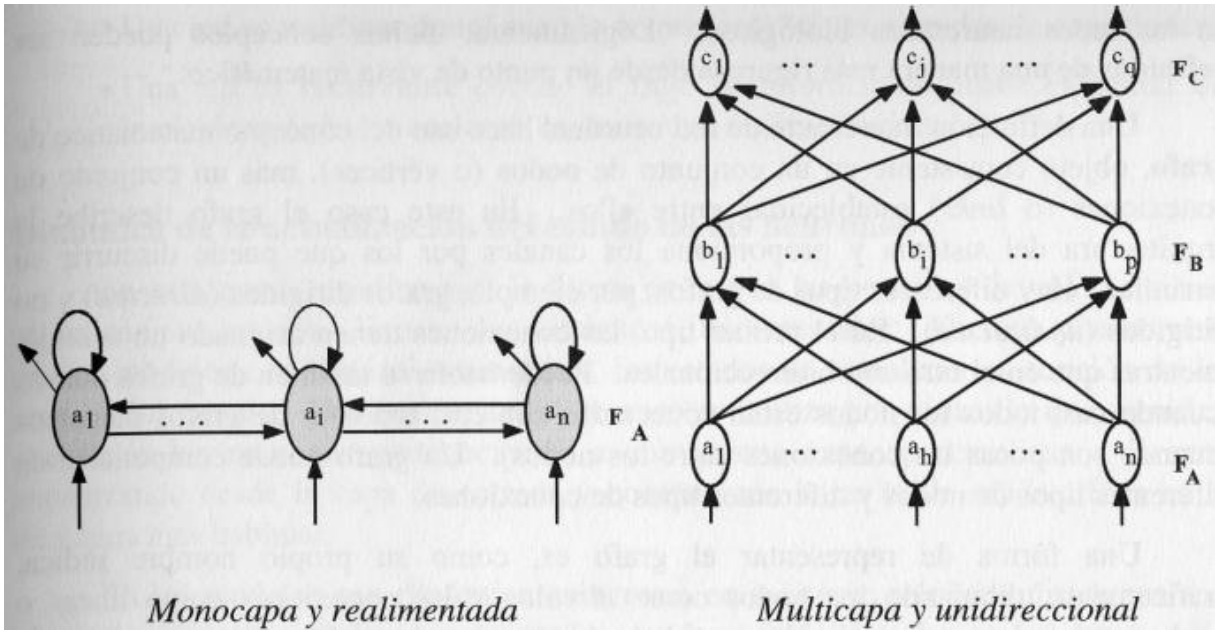


Figura 2.4.4.1 Red monocapa realimentada y multicapa unidireccional

Para formar la red se apilan las capas dando lugar a estructuras complejas con capa de entrada, capa de salida y capas ocultas.

Otro elemento importante que está en las redes neuronales es el aprendizaje, este concepto se refiere a cómo los pesos y las conexiones entre las neuronas se ajustan a medida que la red aprende de los datos de entrenamiento, de forma que se adapta y mejora su rendimiento según recibe más datos de entrenamiento. Hay varios tipos de aprendizaje, pero se van a describir sólo lo más relevantes:

- **Aprendizaje supervisado:** Tiene por objetivo crear una función que pueda predecir el valor numérico o etiqueta de un objeto de entrada, para ello la red recibe el conjunto de datos de entrenamiento etiquetados. La red aprende de estos datos y trata de predecir entradas nuevas no vistas.

- **Aprendizaje no supervisado:** Este tipo de aprendizaje se ajusta a las observaciones, no a un conocimiento previo como en el caso del aprendizaje supervisado. La red se

enfrenta a un conjunto de datos no etiquetados, es decir, desconoce la respuesta correcta por lo que intenta encontrar patrones en los datos por sí misma.

- **Aprendizaje reforzado:** Es un sistema que aprende a base de iteraciones con fallo y error. No hay un agente que guie el aprendizaje, sino que aprende a comportarse con un entorno de recompensas y castigos. Tiene por objetivo maximizar las señales de recompensa para tomar las mejores decisiones posibles.

2.4.5. Redes Neuronales Convolucionales (CNN)

Una vez explicado qué es una red neuronal y los tipos de aprendizaje se va a profundizar un poco más en un tipo de red en concreto, la Red Neuronal Convolucional o Convolutional Neural Network (CNN por sus siglas en inglés).

Es un tipo de red con aprendizaje supervisado que procesa sus capas imitando el ojo humano para identificar las características de su entrada permitiendo identificar objetos, es decir, son una Redes neuronales artificiales que están diseñadas para trabajar principalmente con imágenes. Estas redes son parte del área de Aprendizaje Profundo (de ahora en adelante llamado DL por sus siglas en inglés 'DeepLearning').

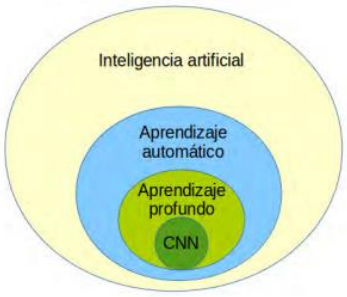


Figura 2.4.5.1 Organización estructural de la IA

Cabe destacar que las CNN hoy en día están entre los sistemas con mejores resultados en el campo de reconocimiento de patrones, extendiendo su uso a imágenes, vídeo, voz y audio.

Las CNN cuentan con capas convolucionales, que se entienden como un conjunto de filtros llamados campos receptivos, las cuales se ajustan para la extracción de características. Una de las diferencias con las ANN está en la conexión de las neuronas; una ANN conecta todas las

neuronas de las capas (Fully Connected), en cambio las CNN comparten neuronas a través de los filtros que permiten extraer información.

El principio de las CNN es que una imagen se puede descomponer en elementos clave y cada capa va aprendiendo con distintos niveles de abstracción. Por ejemplo, una capa aprende sobre aristas, otras sobre patrones más básicos y así sucesivamente hasta aprender patrones complejos (Figura 2.4.9).

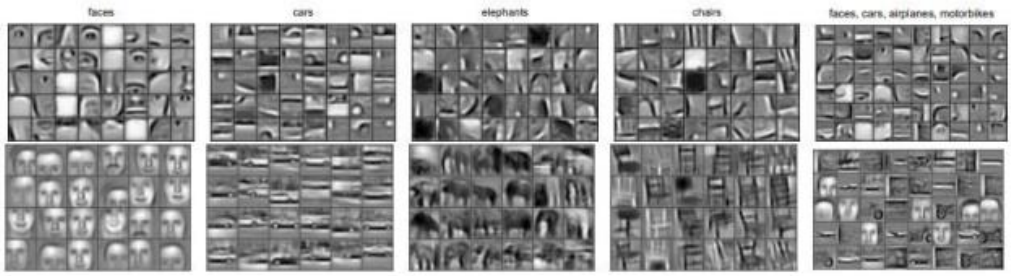


Figura 2.4.5.2 Niveles de abstracción de una CNN

Las CNN tiene una arquitectura formada por tres tipos de capas: convolucional, agregación ("polling"), densa, de activación y puede incluir capas de normalización y abandono ("dropout") según las necesidades del modelo.

- **Capa convolucional:** Su tarea es detectar características o rasgos visuales en las imágenes y, gracias a ello, es capaz de reconocer esa característica en cualquier otro punto de la imagen, permitiendo que pueda aprender de forma eficiente conceptos cada vez más complejos. Para la aplicación de estas capas es común usar filtros o kernels, que recorren la imagen generando mapas de características.

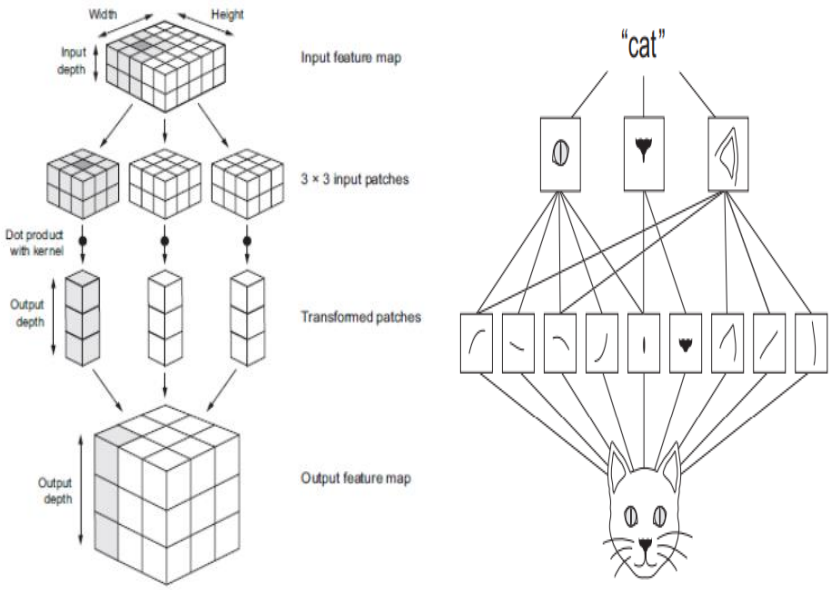


Figura 2.4.5.3 Capa convolucional y ejemplo

- **Capa de agrupación o Pooling:** Esta capa busca condensar la información contenida en la capa convolucional. Para realizar esta tarea hay varias formas, algunas de ellas son: agrupación por máximo ("max-pooling"), por mínimo ("min-pooling") y agrupación por promedio ("average-pooling") (ver figura 2.4.5.4 con un ejemplo de cada uno). Este trabajo es importante para reducir el coste de realizar el aprendizaje, reduciendo el input sin perder información espacial en el proceso.

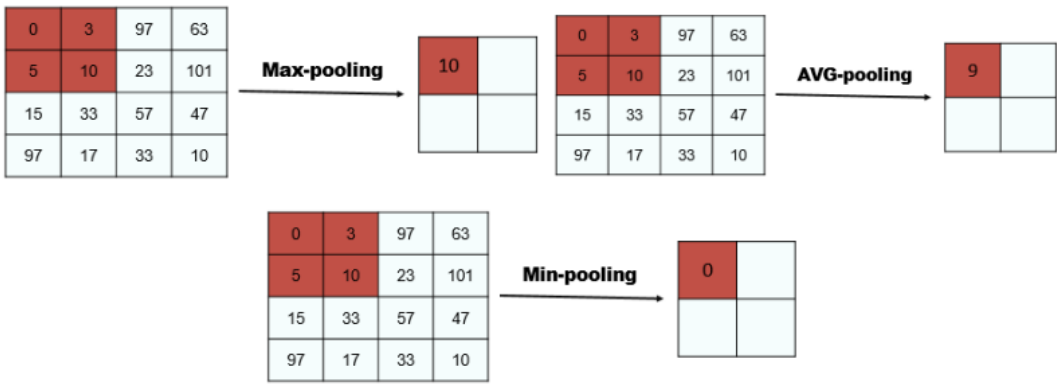


Figura 2.4.5.4 Tipos de agrupación o pooling

- **Capa densa:** Conecta las neuronas de una capa con los elementos de la siguiente y se usa para clasificar las imágenes de diferentes categorías por entrenamiento. Son

similares a las capas de una red neuronal tradicional y son las que se encargan de la clasificación final.

- **Capa de activación:** Es la que define la función de activación de un nodo, la cual define la salida de dicho nodo en función de unas entradas (ver figura 2.4.3.3 para ver algunas de las funciones de activación más usadas). Para este TFG se va a hacer uso de la función de activación softmax, que permite realizar una clasificación con varias opciones calculando la exponencial de cada elemento y luego normaliza cada elemento por la suma de todas las exponenciales, resaltando los valores más grandes y creando una distribución de probabilidades para poder elegir la de mayor probabilidad.

- **Capa de normalización y abandono ("Dropout"):** La capa de normalización busca evitar la dispersión del gradiente, para ello se normaliza. Generalmente, las CNN normalizan por lotes ("Batch") ya que la neurona recibe la entrada de todas. Por otra parte, se tiene la capa de Dropout, la cual se usa para evitar el sobre aprendizaje. Esta técnica consiste en que algunas neuronas dejan de trabajar aleatoriamente durante la fase de entrenamiento, para forzar a las que quedan a aprender en lugar de limitarse a memorizar, con lo que puede clasificar imágenes fuera de los datos del conjunto de entrenamiento.

Hay que añadir que la principal diferencia entre capas de convolución y densas es que las capas densas aprenden patrones globales de la imagen, en cambio las capas de convolución aprenden patrones locales.

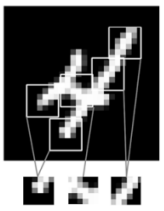


Figura 2.4.5.5 Imagen que se separa en patrones locales

2.5. OpenCV y mediapipe

2.5.1. OpenCV (Open Source Computer Vision Library)

Es una librería ampliamente usada en el tratamiento digital de imágenes y aprendizaje automático. Incluye una gran variedad de algoritmos para la visión artificial, lo que la hace muy útil para el caso de un clasificador de imágenes.

Esta librería se usó por primera vez en 1999 (en su versión alfa) y desde entonces ha ido aumentando exponencialmente su uso hasta convertirse en la biblioteca más popular de visión artificial. Entre algunos de sus usos están: Detección de movimiento, reconocimiento de objetos, o reconstrucción 3D a partir de imágenes, robótica avanzada, etc.

Además, cuenta con grandes ventajas, como puede ser: que es de libre uso tanto para fines comerciales o de investigación, tiene soporte multiplataforma pudiendo ser usada en Linux/Mac/ Windows/Android y para distintas arquitecturas de hardware como x86, x64, ARM o incluso, pese a ser desarrollada en C++, tiene soporte para Python/Java/Matlab/Octave/Javascript. También cuenta como una amplia documentación que se actualiza constantemente.

Según Bradski y Kaehler (2008), OpenCV se estructura de cinco componentes principales:

- Algoritmos de procesamiento de imágenes y de visión por ordenador en nivel básico y superior.
- Uso de ML que incluye clasificadores estadísticos y herramientas de agrupación.
- HighGUI contiene rutinas y funciones de entrada y salida para almacenar y/o cargar videos e imágenes.
- CXCore, contiene las estructuras básicas y algoritmos, apoyo XML, y funciones gráficas.
- CvAux, que contiene algunos algoritmos experimentales.

2.5.2. MediaPipe

Con la creación de las cámaras fotográficas y la extensión de su uso, la tecnología ha ido mejorando cada vez más, incluso llegando a hacer uso de inteligencia artificial para dar apoyo a la captación de imágenes o reproducciones de movimientos. Para ello, Google ha creado una biblioteca capaz de cubrir una amplia variedad de funciones, entre ellas, para el seguimiento y/o reconocimiento de partes del cuerpo como pueden ser las manos, rostros…

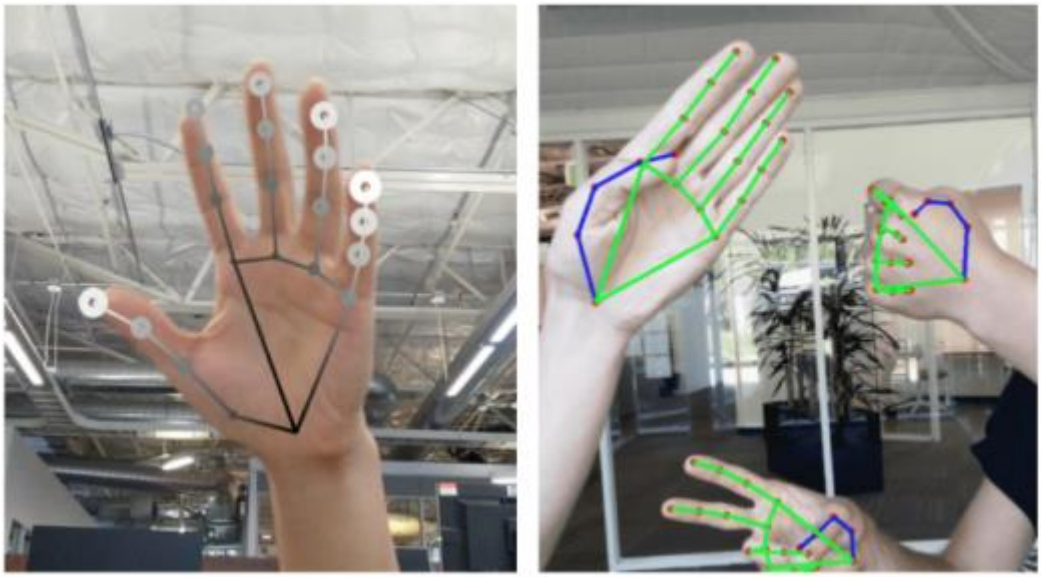
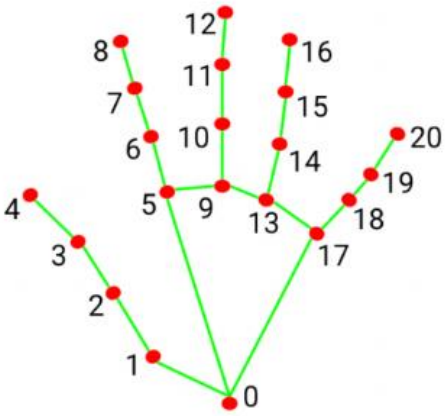


Figura 2.5.2.1 MediaPipe Hand tracking

Como se puede observar en la Figura 2.5.2.1, en el caso del reconocimiento de manos, esta librería hace un mapa de puntos que se asignan a distintos segmentos de la mano. En la figura 2.5.2.2 se puede ver cómo están nombrados cada uno de los 20 puntos que hay.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figura 2.5.2.2

La función MediaPipe Hand Landmarker, es la que se encarga de detectar las manos y generar los puntos de referencia, permitiendo al usuario referenciarse a dichos puntos y trabajar sobre ellos. Esta función opera con datos de imágenes haciendo uso de modelos de Machine Learning (ML) y aprendizaje automático pre-entrenados para simplificar el desarrollo de aplicaciones que usen análisis de video y detección de objetos en tiempo real, en este caso de las manos.

2.6. Raspberry pi 4 y su uso en el proyecto

Actualmente, hay una gran variedad de microprocesadores que se pueden utilizar para construir proyectos. Uno muy popular es el microcontrolador Arduino, que es muy fácil de usar. Sin embargo, hay otras opciones que se pueden asemejar más a un ordenador que el Arduino, una de estas opciones es la Raspberry Pi.

Para el proyecto objeto de esta memoria, se ha decidido hacer uso de una Raspberry pi 4 model B (2018), por su uso extendido, alta fiabilidad y relativamente bajo costo.

Esta placa microordenador fue desarrollada en Inglaterra por la Fundación Raspberry Pi, la cual se dedica a promover la enseñanza de ciencias de la computación. En 2011 se publicó el primer modelo de Raspberry Pi, y según la propia página oficial, se trata de un ordenador de placa reducida de bajo costo basado en la arquitectura ARM.

Aparte del miniordenador, la fundación Raspberry proporciona un sistema operativo propio gratuito, este se llama Raspbian, y es el sistema operativo oficial de esta placa.

Desde el lanzamiento del primer modelo se han lanzado ya varias generaciones de Raspberry Pi, mejorando sus prestaciones cada vez más manteniéndose en un rango de costo accesible para los usuarios.

La Raspberry Pi 4 está basada en el chip Broadcom BCM2711, algunos de los elementos más importantes de los que se conforma son:

- 4 núcleos de ARM Cortex-A72 (ARM V8) de 64-bit en SoC (System On a Chip).
- Dispone de varias versiones de memoria RAM, en este proyecto se usa de 2GB.
- IEEE 802.11 ac Wireless, para la conexión a WiFi.
- Bluetooth 5.0.

- BLE.
- Gigabit Ethernet.
- 2 puertos USB 2.0.
- 2 puertos USB 3.0.
- 40 pin GPIO compatible con versiones precias
- 2 puertos microHDMI
- Ranura MicroSD

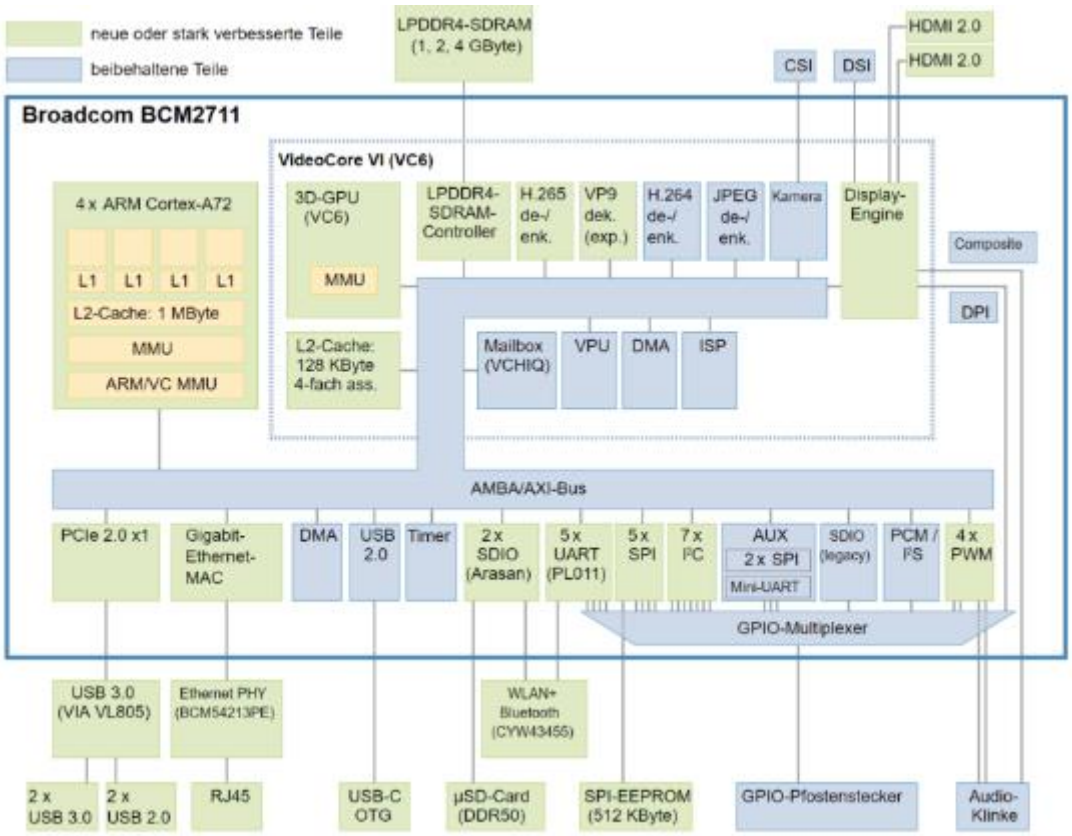


Figura 2.5.2.3 Diagrama de bloques del chip Broadcom BCM2711

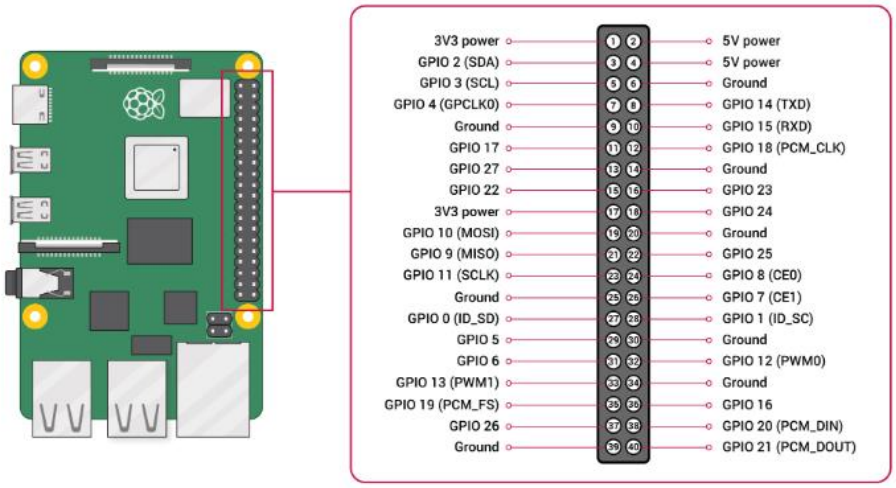


Figura 2.5.2.4 Esquema GPIO Raspberry Pi 4

Algunas ventajas y desventajas de la Raspberry Pi en comparación con otras placas son:

- **Ventajas**
  - Ofrece un muy buen rendimiento a un precio asequible.
  - Cuenta con una gran comunidad de usuarios y desarrolladores que comparten conocimientos, proyectos y soluciones. Por ello el usuario cuenta con un gran soporte y recursos.
  - Cuenta con muchos periféricos y accesorios que facilitan la personalización de proyectos.
  - La Raspberry Pi se puede usar en una gran variedad de aplicaciones, haciéndola una solución muy versátil.
- **Desventajas**
  - Pese a dar un buen rendimiento, en comparación a placas que son mucho más costosas, es un rendimiento limitado.
  - Se calienta más que otras placas y otras versiones anteriores de la misma.
  - Su precio es asequible pero generalmente no incluye elementos como fuente de alimentación o tarjeta microSD con lo que hay que comprarlo a parte.
  - Pese a su mejora gráfica con respecto a versiones anteriores, sigue estando limitada en aplicaciones con un alto rendimiento gráfico.

3. DESARROLLO DEL TFG

3.1. Diseño del sistema

3.1.1. Descripción de los programas desarrollados

Este proyecto consta de 3 programas:

- **Creación del dataset:** Para ello el usuario introduce por teclado las categorías, el número de fotos que se desea por categoría, crea las carpetas para cada categoría y para finalizar, principalmente con ayuda de las librerías OpenCV y mediapipe, toma las fotos del área de interés y se guardan en una dirección especificada.

- **Entrenamiento:** Es el que se encarga de cargar las fotos hechas por el anterior, realiza una adecuación de los datos, define el modelo que se va a usar para clasificar y entrena el modelo. Una vez entrenado el modelo se guarda para usarlo en el programa de clasificación.

  Este programa hace uso de las librerías específicas TensorFlow, sklearn y OpenCV.

- **Clasificación:** Este programa carga el modelo ya entrenado, con el cual va a clasificar. Una vez cargado va a tomar una foto del área de interés, la preprocesa para poder adecuarla al modelo y hace que el modelo la clasifique, imprimiendo su predicción.

3.1.2. Arquitectura del sistema (hardware y software)

En este apartado se va a explicar la arquitectura de sistema que se ha empleado para la creación del clasificador de imágenes. La arquitectura abarca tanto el Hardware como el Software utilizados en el proyecto.

3.1.1.2 Hardware

El sistema se basa en el uso del microordenador Raspberry pi 4 model B (2018), a través del cual se va a ejecutar el programa de clasificación ya que ofrece potencia suficiente para el cometido. Además, se va a hacer uso de una cámara Logitech C920 HD Pro para capturar la imagen y un display OLED SSD 1306 para que imprima la predicción del clasificador de imágenes.

3.1.1.2 Software

El software se basa en el uso de diversas herramientas y bibliotecas. A continuación, se detallan las más importantes:

- **Sistema operativo:** Se ha elegido el Raspbian, que es el sistema operativo oficial que puede ser descargado directamente desde la página oficial de la empresa Raspberry.
- **Python:** Es el lenguaje de programación principal que se ha usado en el desarrollo de este proyecto. Este lenguaje proporciona una gran variedad de herramientas, bibliotecas y guías para el aprendizaje automático, procesamiento de imágenes y manejo de la placa Raspberry pi.
- **OpenCV:** Como se ha indicado antes esta es la librería más popular que se encarga de gestionar la visión artificial.
- **TensorFlow y Keras:** Estas dos bibliotecas son específicas y optimizadas para el aprendizaje automático y redes neuronales, por lo que son las que dan forma al programa de entrenamiento. TensorFlow permite la implementación de redes neuronales de forma eficiente y optimizada, mientras que Keras facilita la construcción y entrenamiento del clasificador.
- **Mediapipe:** También mencionada antes, es la biblioteca desarrollada por Google que simplifica la detección de la posición de las manos y ayuda a extraer el área de interés.

3.2. Implementación del programa de creación del dataset

3.2.1. Detalles de la estructura y organización del dataset

Para el diseño de este trabajo, inicialmente se planteó usar un dataset ya hecho por otro usuario, para ello se hizo una búsqueda en el popular repositorio Kaggle y se eligieron distintos datasets para probarlos y ver el rendimiento del clasificador entrenado. Los distintos problemas con los datasets usados se especificarán más adelante.

Puesto que los datasets descargados de kaggle daban distintos fallos, al final se optó por crear un dataset desde cero y para ello se creó un programa específico que, en resumen, accede a la cámara, dibuja un cuadro alrededor de la mano y usando una función, guarda en una dirección especificada.

Ahora se va a explicar las librerías usadas y después se va a explicar en detalle el programa.

- **Librerías**
    - **Mediapipe:** Es una librería de código abierto desarrollada por Google que se va a usar para identificar las manos y su posición central, que se va a usar de referencia para dibujar el cuadro.
    - **Os:** Esta es una librería estándar de Python que permite la interacción con el sistema operativo. Esta librería se usa para crear/renombrar archivos y carpetas, obtener rutas sobre dichas carpetas archivos, etc.
    - **Cv2:** Abreviación de la librería OpenCV, como se ha explicado antes, se usa para la visión artificial y procesamiento de imágenes. Esta es la librería que va a acceder a la cámara.
    - **Matplotlib:** Se encarga de la representación de datos, permitiendo crear gráficos, diagramas, entre otros. En este caso se usa para representar imágenes y datos.
    - **Numpy:** Librería fundamental para la computación numérica en Python que proporciona funciones para realizar cálculos numéricos, tratamiento de matrices y arreglos multidimensionales.
    - **Sutil:** Otra librería estándar de Python que proporciona funciones más avanzadas para operaciones de manipulación de archivos. En este programa se usa para limpiar la carpeta del dataset.
    - **Threading:** Librería que proporciona herramientas para la programación concurrente. En este proyecto se usa para ejecutar simultáneamente una función y el programa principal.

```
[1]: # Librerias que se van a usar para crear el data set

     import mediapipe as mp            # Libreria que identifica las manos
     import os                         # Libreria para gestión de archivos
     import cv2                        # OpenCV
     import matplotlib.pyplot as plt   # Librerá para representación de datos e imágenes
     import numpy as np                # Libreria para operaciones numéricas
     import shutil                     # Librería que ayuda a eliminar todo lo que hay en la carpeta donde se van a guardar las fotos
     import time                       # Librería que detiene una tarea para poder cambiar la posición de la mano entre foto y foto
     import threading                  # Librería para ejecutar la captura de imagenes en segundo plano, asi no se para la imagen
```

Figura 3.1.1.1.1 Librerías para creación de data set

Una vez introducidas las librerías se va a desarrollar el programa, se inicializan algunas funciones. La variable “capture” va a gestionar la apertura y cierre de la cámara, mientras que “clase_manos” y “manos” se usa para trabajar con la librería mediapipe.

```
# Inicialización de la cámara y detección de manos
capture = cv2.VideoCapture(0)                          # Abro la cámara
clase_manos  =  mp.solutions.hands                     # accedo a las herramientas de mediapipe para reconocimiento de manos
manos = clase_manos.Hands()
```

Figura 3.1.1.1.2 Inicialización

Para permitir un uso más genérico del programa en lugar de introducir un vector fijo conteniendo las clases que se van a categorizar, en este caso las letras del abecedario a-z (sin la ñ), se ha creado un trozo que se va a encargar de preguntar al usuario cuales son las categorías (una a una) que se desean para crear las carpetas correspondientes donde se van a guardar las fotos y una vez se hayan introducido todas las categorías deseadas, el usuario debe escribir “fin”. El uso de la espera de “fin” se hace para no limitar a un número fijo las categorías a crear.

```
[15]: # Se pide al usuario que introduzca los elementos de los que va a querer tomar fotos, en este caso el abecedario y números en lenguaje de signos

      classes = []                    # Vector donde se van a guardar las clases de las que se van a tomar las fotos

      while True:                     # Bucle para rellenar el vector classses[]
          dato = input("¿Introduzca los datos a tomar?(introducir 'fin' cuando se haya acabado: ")    # Espera que el usuario introduzca el nombre de la clase y pulse 'enter'

          if dato == 'fin':           # Bandera que indica que se ha acabado de completar el vector classes[]
              break
          else:
              classes.append(dato)    # Añade al vector el dato introducido por el usuario
              print("classes = ", classes) # Muestra cómo esta el vector hasta el momento

      print("los datos que se van a tomar son: ", classes)   # Cuando se ha acabado de introducir datos se muestra el resultado final del vector
```

Figura 3.1.1.1.3 Definición de categorías

Como se ha mencionado, para este proyecto el vector contendrá las letras de la a-z de abecedario inglés y se va a denominar “classes”.

```
classes =  ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
¿Introduzca los datos a tomar?(introducir 'fin' cuando se haya acabado:  fin
```

Figura 3.1.1.1.4 Categorías a usar en este proyecto

Una vez definidas las categorías que va a tener el programa, se procede de forma similar, preguntando al usuario cuántas fotos quiere el usuario para cada una de las categorías. En este

caso se ha limitado a que se introduzca un número de 2 dígitos de forma arbitraria, podría modificarse a cualquier otro límite o incluso quitarlo.

```
[4]: # Se pide al usuario que introduzca el número de fotos que se quiere tomar para cada muestra

     while True:                          # Bucle que pide al usuario el numero de fotos
         num_fotos = input("¿Cuántas fotos quiere tomar?: ")   # Espera que el usuario introduzca un número entre 0-99 y pulse 'enter'

         try:                                                  # Comprueba que se haya introducido un número entre 0-99
             num_fotos = int(num_fotos)
             if num_fotos >= 0 and num_fotos <= 99:
                 break
             else:
                 print("Error: Introduzca un número entre 0-99")
         except ValueError:
             print("Error: Introduzca un número entero válido") # Fin comprobación

     print("Ha seleccionado tomar", num_fotos, "fotos")        # Se muestra el número que de fotos que se ha elegido tomar
     num_fotos = int(num_fotos)                                # Convierto el dato a int para poder trabajar con él más adelante
     ¿Cuántas fotos quiere tomar?:  30
     Ha seleccionado tomar 30 fotos
```

Figura 3.1.1.1.5 Número de fotos que se van a tomar

Para el data set final se han usado 70 fotos de un dataset descargado de Kaggle, y para mejorar su comportamiento se han añadido 30 fotos de la mano de un hombre y 30 fotos de la mano de una mujer.

A continuación, limpia todo lo que haya en la carpeta donde van a estar las categorías, eliminando así categorías de programas anteriores o duplicados evitando posibles problemas. Para este proyecto se va a tener en la carpeta llamada 'my_dataset' varias carpetas con las categorías y dentro de cada carpeta de categoría estarán las fotos nombradas con el nombre de la categoría seguida de un número.

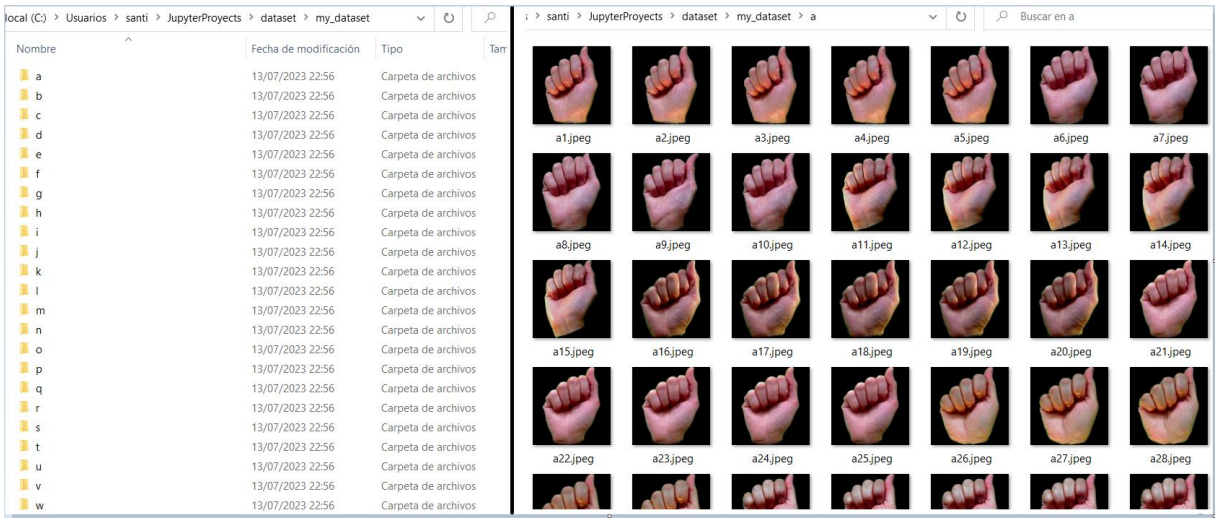


Figura 3.1.1.1.6 Contenido de la carpeta "my_dataset" y contenido de una de las carpetas

```
[12]: # Limpio lo que haya en la carpeta en la cual se va a guardar el data set para no mezclar con fotos antiguas

folder_path = 'my_dataset'                          # Dirección donde se va a guardar el dataset

# Eliminar todos los archivos dentro de la carpeta
for filename in os.listdir(folder_path):
    file_path = os.path.join(folder_path, filename)
    try:
        if os.path.isfile(file_path) or os.path.islink(file_path):    # Si es un archivo/link se borra
            os.unlink(file_path)
        elif os.path.isdir(file_path):                                # Si es una carpeta se borra
            shutil.rmtree(file_path)
    except Exception as e:
        print('Error al borrar %s. Razón: %s' % (file_path, e))       # Si no se ha podido borrar alguna de las carpetas/archivos da mensaje de error
```

Figura 3.1.1.1.7 Limpieza de la carpeta “my_dataset”

Una vez limpia la carpeta y definidas las categorías que se desean, se van a crear las carpetas que, asignadas a cada una de las categorías, para ello se hace uso de un bucle que recorre el vector de “classes”, extrayendo la categoría y creando una carpeta con su nombre.

```
[13]: # Se crean las carpetas para guardar cada una de las clases que se introducieron por teclado en el vector classes[]

# Ruta de la carpeta que contiene todas las carpetas de imágenes
path = 'C:\\Users\\santi\\JupyterProyects\\dataset\\my_dataset_2'       # Dirección donde se van a crear las carpetas

for foldername in classes:                                             # Bucle que recorre el vector classes[]
    class_name = os.path.join(path, foldername)                        # Llama a cada carpeta que se va a crear como el vector con el que se va relacionar
    os.mkdir(class_name)                                               # Crea la carpeta
    print("se ha cerado la carpeta para: ", foldername)                # Mensaje de confirmación de que se han creado las carpetas

se ha cerado la carpeta para:  a
se ha cerado la carpeta para:  b
se ha cerado la carpeta para:  c
se ha cerado la carpeta para:  d
se ha cerado la carpeta para:  e
se ha cerado la carpeta para:  f
se ha cerado la carpeta para:  g
se ha cerado la carpeta para:  h
se ha cerado la carpeta para:  i
se ha cerado la carpeta para:  j
se ha cerado la carpeta para:  k
se ha cerado la carpeta para:  l
se ha cerado la carpeta para:  m
se ha cerado la carpeta para:  n
se ha cerado la carpeta para:  o
se ha cerado la carpeta para:  p
se ha cerado la carpeta para:  q
se ha cerado la carpeta para:  r
se ha cerado la carpeta para:  s
se ha cerado la carpeta para:  t
se ha cerado la carpeta para:  u
se ha cerado la carpeta para:  v
se ha cerado la carpeta para:  w
se ha cerado la carpeta para:  x
se ha cerado la carpeta para:  y
se ha cerado la carpeta para:  z
```

Figura 3.1.1.1.8 Creación de las carpetas relacionadas a las categorías

En este punto, se tiene creadas las carpetas, definidas las categorías y el número de fotos que se va a hacer para cada una, así que se procede con la toma de fotos. Para ellos se ha creado una función llamada “tomar_fotos” que recibe 2 argumentos:

- **Folder_name:** Que contiene la dirección asociada a una categoría.
- **Num_fotos:** Que como su nombre indica, contiene el número de fotos que se van a hacer para esa categoría.
- **X1, X2, Y1, Y2:** Coordenadas que forman un cuadro con el área de interés.

Función “tomar_fotos”:

Esta función primero comprueba que exista la carpeta de la categoría correspondiente. Una vez comprobado esto se procede a entrar en un bucle el cual primero define el nombre con el que se va a guardar la foto (recordemos es el nombre de la carpeta seguido de un número) y se concatena la dirección con el nombre. Por ejemplo, si tenemos la dirección de carpeta “C:// users/Project/my_dataset/a” y el nombre que queremos es “a1” ya que es la primera foto, al concatenar nos queda “C:// users/Project/my_dataset/a/a1.jpeg”

```
        photo_name = f"{folder_name}{i+101}.jpeg"          # Nombre con el que se va a guardar la foto (el nombre de la carpeta mas un numero)
        photo_path = os.path.join(dir_path, photo_name)    # Dirección de la foto (incluye el nombre)
```

Figura 3.1.1.1.9 Definición de la dirección para guardar la imagen

Con la dirección y nombre definidos, se procede a hacer una captura de lo que se visualiza en la Cámara con la instrucción “capture.read” de OpenCV.

La función termina haciendo uso de las coordenadas que se han introducido, recortando la imagen según el área de interés ROI (Region of interest) y guardándola según la dirección y nombre previamente indicados. Se ha añadido un tiempo de espera de 1 segundo para que el usuario pueda realizar pequeñas modificaciones en la posición de la mano, evitando así que se hagan todas las fotos seguidas y por tanto, que sean demasiado parecidas.

```
        ret, frame = capture.read()                    # capturo la imagen

        # Recortar la ROI
        roi = frame[Y1:Y2, X1:X2]                      # Recorte de la imagen quedando solamente el recuadro que contiene la mano

        cv2.imwrite(photo_path, roi)                   # Guardo la imagen
        print("se ha tomado la foto: ", i)
        time.sleep(1)                                  # espera x segundos antes de tomar la siguiente foto
```

Figura 3.1.1.1.10 Recorte de área de interés y guardado de la imagen recortada

Bucle

Antes de empezar con el bucle que se encarga de extraer las coordenadas del área de interés ROI, se hace una inicialización en la que se habilita la captura de imagen y se instancian elementos para el uso de la librería mediapipe.

```
# Inicialización de la cámara y detección de manos
capture = cv2.VideoCapture(0)                          # Abro la cámara
clase_manos  =  mp.solutions.hands                     # accedo a las herramientas de mediapipe para reconocimiento de manos
manos = clase_manos.Hands()
```

Figura 3.1.1.1.11 Inicialización e instancia

Para extraer el área de interés se ha hecho uso de un bucle que constantemente hace una captura de la imagen con la que se va a hacer un tratamiento para extraer los puntos que delimitan dicha área y posteriormente se dibuja un rectángulo para que el usuario pueda verla. Esa área, ayuda a que se pueda asegurar de que mantiene la mano dentro del cuadro para un mejor resultado en la creación del dataset.

Se optó con la creación de un cuadro fijo en el área de interés ya que desarrollar un sistema que tenga en consideración la cercanía/lejanía de la mano sería muy complicado para el ámbito de este proyecto, y el cuadro es una alternativa efectiva.

En este bucle lo primero que se hace es hacer la captura dela imagen antes mencionada y se cambia el formato de color de BGR (Blue-Green-Red) a RGB (Red-Green-Blue), ya que OpenCV por defecto trata las imágenes en formato BGR y mediapipe espera imágenes con formato RGB.

```
ret, frame = capture.read()                    # capturo la imagen
color = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)    # Inicio procesado de la imagen para recortar solamente la mano
```

Figura 3.1.1.1.12 Captura y adecuación de formato

Ahora, con la imagen en el formato correcto, se obtendrá el mapa de puntos utilizando la biblioteca mediapipe. Para ello se utiliza "manos.process", esta función es la que pasa la imagen por el modelo de detección y seguimiento de manos desarrollado por medipipe, creando un mapa de puntos que se almacena en el atributo ".multi_hand_landmarks".Véase en la imagen 3.1.1.1.13 que, además, se crea un vector llamado “posiciones”, dicho vector va a contener elementos de 3 datos:

- El id del punto, el cual es un número entre 0-20 (ver figura 2.5.2.2)
- La coordenada x de dicho punto
- La coordenada y de dicho punto

```
resultado = manos.process(color)
posiciones = []
```

Figura 3.1.1.1.13 Introducción en modelo de mediapipe

En este momento se tiene en la variable “resultados” la imagen de la mano con el mapa de puntos, sin embargo, las coordenadas que nos devuelve en x e y están normalizadas. Por tanto, para obtener las coordenadas reales se tiene que multiplicar por el ancho y alto real de la imagen. Para conseguir eso se ha usado un pequeño sub-bucle que recorre cada punto: primero guarda las dimensiones de la imagen, luego multiplica esas dimensiones por la coordenada normalizada y guarda el id junto con las coordenadas reales en el vector “posiciones”.

```
if resultado.multi_hand_landmarks:
    for mano in resultado.multi_hand_landmarks:
        for id, lm in enumerate(mano.landmark):
            alto, ancho, c = frame.shape
            corx, cory = int(lm.x*ancho), int(lm.y*alto)
            posiciones.append([id,corx,cory])
```

Figura 3.1.1.1.14 Creación vector con el mapa de puntos

Arbitrariamente se ha elegido el punto 9 (ver figura 2.5.2.2) como el más próximo al centro de la mano, y con esa referencia se marcan los extremos de un cuadro alrededor. El área que contiene el cuadro es la región de interés ROI, ya que contiene solamente la mano y elimina el resto de información que no es relevante para el clasificador.

```
if len(posiciones) != 0:
    pto_i9 = posiciones[9]
    ancho, alto = (pto_i9[1]),(pto_i9[2])
    x1,y1 = (ancho-150),(alto-150)
    x2,y2 = (ancho+150), (alto+150)
    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 3)     # Fin procesado de la imagen
```

Figura 3.1.1.1.15 Extracción de región de interés.

Para poder hacer que las fotos se hagan cada segundo mientras que el usuario puede ver en tiempo real la imagen de la cámara, hizo falta añadir una forma de que la función antes definida y el bucle se ejecuten en planos distintos. Esto se consiguió con el uso de la librería threading, la cual implementa una instrucción que ejecuta en segundo plano una función, para ello se introduce como primer argumento el nombre de la función y como segundo argumento lo elementos que espera la función.

```
# Iniciar hilo para tomar las fotos
t = threading.Thread(target=tomar_fotos, args=(class_name, num_fotos, x1, x2, y1, y2))  # Instrucción que ejecuta en segundo plano la función que toma las fotos
t.start()                                                                               # Inicia la ejecución de la línea anterior
```

Figura 3.1.1.1.16 Ejecución en segundo plano

En conclusión, el programa hace emerger una ventana donde el usuario verá un cuadro alrededor de la mano, espera a que se introduzca una señal por teclado (tecla ‘c’) para empezar a hacer las fotos de una categoría, una vez ha acabado de hacer el número de fotos espera que se vuelva a mandar la señal por teclado. Para interrumpir el programa o para confirmar que se ha acabado se debe presionar la tecla ‘q’, con lo que sale del bucle, deja de capturar la imagen y cierra la ventana emergente.

```
# Aquí se va a dibujar un cuadro de dimensiones fijas al rededor de la mano y se recorta la imagen, luego con la función se guardan las fotos dentro de cada carpeta

# Función para tomar las fotos
def tomar_fotos(folder_name, num_fotos, X1, X2, Y1, Y2):                    # Función que toma las fotos, repito el procesado y recorte para actualizar el roi

    dir_path = os.path.join(folder_path, folder_name)        # Dirección de la carpeta del conjunto de fotos (la letra de la que se van a tomar fotos)
    if not os.path.exists(dir_path):                         # Si la carpeta no existe no existe devuelve mensaje de error
        print("ERROR: la carpeta para ", folder_name, " no se ha creado correctamente")
        return
    for i in range(num_fotos):                               # Bucle que toma las fotos
        photo_name = f"{folder_name}{i+101}.jpeg"            # Nombre con el que se va a guardar la foto (el nombre de la carpeta mas un numero)
        photo_path = os.path.join(dir_path, photo_name)      # Dirección de la foto (incluye el nombre)

        ret, frame = capture.read()                          # capturo la imagen

        # Recortar la ROI
        roi = frame[Y1:Y2, X1:X2]                             # Recorte de la imagen quedando solamente el recuadro que contiene la mano

        cv2.imwrite(photo_path, roi)                          # Guardo la imagen
        print("se ha tomado la foto: ", i)
        time.sleep(1)                                         # espera x segundos antes de tomar la siguiente foto
```

Figura 3.1.1.1.17 Función completa

```
# Inicialización de la cámara y detección de manos
capture = cv2.VideoCapture(0)                                # Abro la cámara
clase_manos  =  mp.solutions.hands                           # accedo a las herramientas de mediapipe para reconocimiento de manos
manos = clase_manos.Hands()

# Variables para el rectángulo de la zona de interés
x1, y1, x2, y2 = 0, 0, 0, 0                                  # Variables donde se van a guardar las coordenadas del cuadro que contiene la mano
z = 0                                                        # Inicio contador a cero que se va a usar en classes[z] para recorrer el vector
while (capture.isOpened()):                                  # Bucle para visualizar lo que se observa en la cámara

    ret, frame = capture.read()                              # Capturo la imagen
    color = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)       # Inicio procesado de la imagen para recortar solamente la mano
    resultado = manos.process(color)
    posiciones = []
    if resultado.multi_hand_landmarks:
        for mano in resultado.multi_hand_landmarks:
            for id, lm in enumerate(mano.landmark):
                alto, ancho, c = frame.shape
                corx, cory = int(lm.x*ancho), int(lm.y*alto)
                posiciones.append([id,corx,cory])
            if len(posiciones) != 0:
                pto_i9 = posiciones[9]
                ancho, alto = (pto_i9[1]),(pto_i9[2])
                x1,y1 = (ancho-150),(alto-150)
                x2,y2 = (ancho+150), (alto+150)
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 3)     # Fin procesado de la imagen

    cv2.imshow('webCam',frame)                                   # Muestro la cámara en pantalla

    if cv2.waitKey(1) & 0xFF == ord('c') and z < len(classes):     # Si se presiona la tecla 'c' se procede a tomar las fotos

        #print("Presione 'c' para empezar")
        class_name = classes[z]                                  # Inicializo en la posición 0 del vector que contiene las clases
        print("Tomando fotos para ", class_name)                 # Notifica de qué se están haciendo las fotos

        # Iniciar hilo para tomar las fotos
        t = threading.Thread(target=tomar_fotos, args=(class_name, num_fotos, x1, x2, y1, y2))  # Instrucción que ejecuta en segundo plano la función que toma las fotos
        t.start()                                                                               # Inicia la ejecución de la línea anterior
        z += 1              # Pasa al siguiente elemento del vector clases[z] y sal del if, queda a la espera de que se vuelva a presionar 'c'

    if cv2.waitKey(1) & 0xFF == ord('q') and z >= len(classes) or cv2.waitKey(1) & 0xFF == ord('q'): # Si se han hecho todas las fotos y/o se presiona 'q' sal del bucle
        break


capture.release()           # Suelta la cámara
cv2.destroyAllWindows()     # Cierra la ventana emergente para visualizar la cámara
```

Figura 3.1.1.1.18 Inicialización y bucle completo

3.3. Implementación del programa de entrenamiento del modelo

Este programa es el corazón del proyecto, ya que es el que se encarga de adecuar los datos de entrenamiento, definir el modelo y entrenar la red neuronal para que aprenda a clasificar el lenguaje de signos.

Usa las imágenes que se capturaron en el programa anterior para el entrenamiento y se añadieron 70 fotos de cada letra de un dataset que se obtuvo por la página Kaggle, haciendo un total de 130 fotos por cada letra, 70 de Kaggle, y 60 hechas con el programa anterior (30 de hombre y 30 de mujer).

A continuación, se va a hacer una breve descripción de el uso que se ha dado a las librerías usadas (Figura ---):

- **Os:** Al igual que en el anterior programa se usa para poder acceder a los archivos del ordenador, en este caso ayuda a cargar el dataset y leer las categorías para establecer las etiquetas.
- **OpenCV:** Se usa para la visión artificial, permitiendo cargar las imágenes.
- **Numpy:** Permite trabajar con números, es muy útil ya que para procesar los datos se convierten en arreglos numéricos.
- **Matplotlib, sklearn y seaborn:** Encargadas la visualización de datos y resultados.
- **TensorFlow y Keras:** Como se ha mencionado antes, son bibliotecas muy usadas para el aprendizaje automático, que permiten la creación y gestión de las redes neuronales.

```
# Librerias para cargar los datos
import os
import cv2
import numpy as np

# Librerias para visualizar los datos
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix        # Librerias usadas para hacer la matriz y para hacer el heatmap
import seaborn as sns

# Librerias para el modelo
from tensorflow.keras import utils
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, BatchNormalization
from sklearn.model_selection import train_test_split
```

Figura 3.3.1 Librerías usadas

Puesto que este programa es el que requiere mayor uso de los componentes, para que se pueda entrenar de forma mucho más rápida se ejecuta usando la GPU del ordenador. Puesto

que al inicio no se tenían instalados programas necesarios para permitir el uso de la GPU, se tuvo que hacer una configuración y actualización de distintas dependencias:

- **NVIDIA GPU Drivers:** controladores actualizados correspondientes a la tarjeta gráfica.
- **CUDA Toolkit:** TensorFlow usa esta plataforma de NVIDIA para aprovechar la potencia de la GPU.
- **cuDNN Library:** La librería NVIDIA CUDA Deep Neuronal Network, acelera las redes neuronales y optimiza las operaciones con uso de la GPU, pero necesita que estén instalados CUDA Toolkit y TensorFlow.
- **TensorFlow-GPU:** Es la librería que ayuda a la gestión de las redes convolucionales, aunque a partir de la versión 2.10.0 de TensorFlow se proporciona uso de la GPU.

Figura 3.3.2 Comprobación de que la GPU esté activa

3.3.1. Configuración de la red CNN

La red neuronal va a ser tipo convolucional, para ello se hace uso de un modelo secuencial, es decir que la salida de una capa es la entrada de la siguiente capa.

Todas las capas ocultas tienen la misma estructura:

- Una capa convolucional que aplica un número determinado de filtros, que rellena los bordes para que la salida tenga la misma forma que la entrada (padding='same'), que esperan una imagen de 75x75 píxeles y usa la función de activación ReLu.
- Una capa MaxPooling para agrupar las características que se extraen de la capa convolucional, para ello se ha establecido una ventana de 3x3 píxeles para poder enfatizar en detalles. Una ventana muy grande significaría la pérdida de detalles específicos debido a que con una ventana mayor tiene a extraer características más generales.

- Una capa de normalización, que normaliza los mini-lotes para estabilizar y optimizar el proceso de entrenamiento.
- Capa de Dropout, que sirve para que se apaguen neuronas aleatoriamente forzando a las que no se han apagado a intentar aprender mejor. Esta capa se usa para evitar el sobre aprendizaje.

Para finalizar la red, se usa una capa de aplanamiento (Flatten) que transforma la salida de la capa convolucional, que es un tensor tridimensional, y la convierte en un vector unidimensional. Esto es necesario para conectar la salida de las capas convolucionales con la capa densa que viene a continuación de esta. Dicha capa densa antes de la capa de salida tiene el cometido de introducir no linealidad en la red para mejorar la extracción de datos.

Como capade salida se ha usado una función de activación 'softmax' que es adecuada para problemas de clasificación multiclase (tenemos un total de 26 categorías), esta función trabaja con una función de probabilidad para poder determinar la etiqueta de salida.

```
# Creo el modelo

model = Sequential()

model.add(Conv2D(64, (3, 3), padding='same', input_shape=(75, 75, 1), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(128, (3, 3), padding='same', input_shape=(75, 75, 1), activation='relu')) #128
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Conv2D(256, (3, 3), padding='same', input_shape=(75, 75, 1), activation='relu')) # 256
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(BatchNormalization())
model.add(Dropout(0.8))

model.add(Flatten())
model.add(Dense(1024, activation='relu'))  #1024
model.add(Dense(classes_len, activation='softmax'))
```

Figura 3.3.1.1 Modelo de la red convolucional y sus capas

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 75, 75, 64)        640

max_pooling2d (MaxPooling2D  (None, 25, 25, 64)        0
)

batch_normalization (BatchN  (None, 25, 25, 64)        256
ormalization)

dropout (Dropout)            (None, 25, 25, 64)        0

conv2d_1 (Conv2D)            (None, 25, 25, 128)       73856

max_pooling2d_1 (MaxPooling  (None, 8, 8, 128)         0
2D)

batch_normalization_1 (Batc  (None, 8, 8, 128)         512
hNormalization)

dropout_1 (Dropout)          (None, 8, 8, 128)         0

conv2d_2 (Conv2D)            (None, 8, 8, 256)         295168

max_pooling2d_2 (MaxPooling  (None, 2, 2, 256)         0
2D)

batch_normalization_2 (Batc  (None, 2, 2, 256)         1024
hNormalization)

dropout_2 (Dropout)          (None, 2, 2, 256)         0

flatten (Flatten)            (None, 1024)              0

dense (Dense)                (None, 1024)              1049600

dense_1 (Dense)              (None, 26)                26650

=================================================================
Total params: 1,447,706
Trainable params: 1,446,810
Non-trainable params: 896
```

Figura 3.3.1.2 Resumen de la CNN

3.3.2. Preprocesamiento de imágenes

Antes de hacer el procesado de las imágenes, se realiza un conteo de las carpetas creadas en el anterior programa, esto hace que se genere el vector que contiene las categorías y a modo informativo, se imprime en pantalla el vector, los elementos del vector y el número de fotos que hay para cada categoría.

```
# Hago un conteo de cuantas imagenes hay en cada carpeta

classes = os.listdir(train_path)
print(classes)

print(len(os.listdir(train_path)))
for i in range(len(os.listdir(train_path))):
    print("{0} tiene: ".format(classes[i]),len(os.listdir('C:\\Users\\santi\\JupyterProyects\\dataset\\my_dataset' + "/{0}".format(classes[i]))))
print("Fin conteo")
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
26
a tiene:  130
b tiene:  130
c tiene:  130
d tiene:  130
e tiene:  130
f tiene:  130
g tiene:  130
h tiene:  130
i tiene:  130
j tiene:  130
k tiene:  130
l tiene:  130
m tiene:  130
n tiene:  130
o tiene:  130
p tiene:  130
q tiene:  130
r tiene:  130
s tiene:  130
t tiene:  125
u tiene:  130
v tiene:  130
w tiene:  130
x tiene:  130
y tiene:  130
z tiene:  130
Fin conteo
```

Figura 3.3.2.1 Definir vector de categorías y conteo de imágenes

Con el vector de categorías creado se procede a cargar las imágenes, para esto se crea una función llamada "get_data". Esta función espera la dirección (path) donde se encuentran las carpetas que contienen los datos de entrenamiento. A continuación, se detalla el funcionamiento de la función.

Get_data:

La función empieza definiendo dos vectores, en uno se van a guardar las imágenes y en el otro se van a guardar las etiquetas correspondientes.

```
images = []                                          # Vectores donde guardaren los datos cargados
labels = []
```

Figura 3.3.2.2 Vectores que van a contener las imágenes y las etiquetas

Para cargar los datos se hace uso de dos bucles anidados, el primero se ejecuta tantas veces como categorías existen en la carpeta de los datos de entrenamiento, en este caso 26 veces. El segundo bucle está dentro del anterior, y recorre la carpeta de cada categoría cargando las fotos en una variable haciendo uso de la librería OpenCV, además se redimensiona la imagen para que tenga la forma que el clasificador espera (75x75 píxeles). Tener en cuenta que las imágenes se han cargado en blanco y negro para evitar problemas en cuanto a diferentes tonos de piel.

Para finalizar el segundo bucle, se concatena la imagen que se ha cargado y la etiqueta en los vectores correspondientes. La función devuelve el vector de imágenes con todas las imágenes cargadas y el vector que contiene las etiquetas.

```
for i in range(len(dir_list)):                       # Bucle que carga la imagen
    print("Cargando imagenes de", dir_list[i], "...")
    for image in os.listdir(data_dir + "/" + dir_list[i]):
        img = cv2.imread(data_dir + '/' + dir_list[i] + '/' + image,0)
        img = cv2.resize(img, (75,75))               # Redimensiono la imagen para el modelo
        # img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        images.append(img)                           # Añado la imagen al vector imagenes
        labels.append(i)                             # Añado la etiqueta al vector de etiquetas

return images, labels                                # La función devuelve el vector con las imagenes y las etiquetas
```

Figura 3.3.2.3 Cargar las imágenes y etiquetas e introducirlas en los vectores

Una vez definida la función, solo hay que llamarla y guardar los datos en las variables que se han elegido (X para las imágenes e Y para las etiquetas).

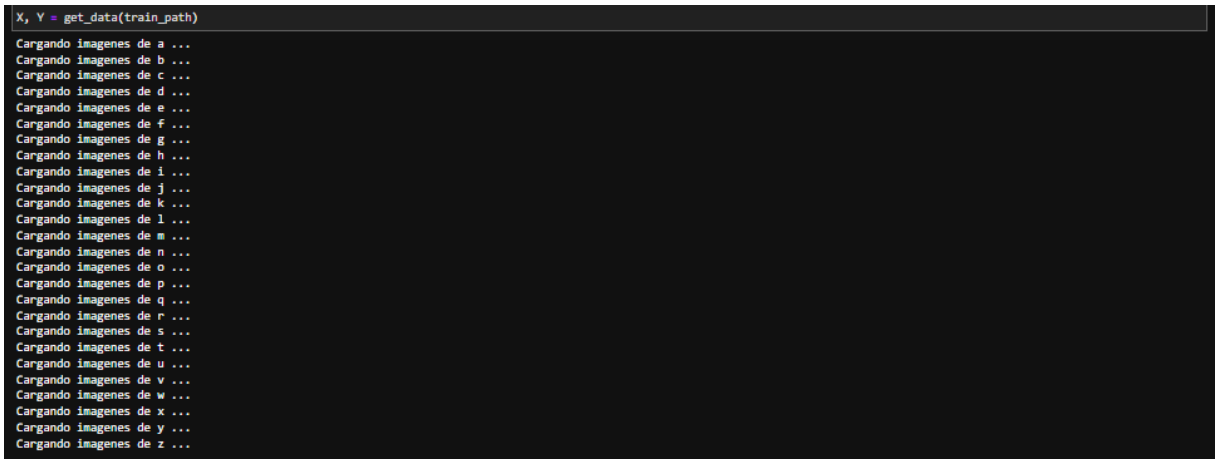


Figura 3.3.2.4 Llamada de la función “get_data”

Luego de llamar a la función se tiene:

- **Variable ‘X’:** Vector que contiene las imágenes con las dimensiones que espera el clasificador.
- **Variable ‘Y’:** Vector que contiene las etiquetas.

Para un mejor funcionamiento se suelen normalizar los datos y, además, se necesita que las etiquetas estén en una matriz para introducirlas al clasificador. Para hacer estas dos cosas se ha creado otra función llamada “preprocess_data”

Preprocess_data:

En esta función primero se normalizan los datos que se tienen en X, pero para ello hay que convertirlo en un arreglo numérico que facilite su manipulación, para ello se hace uso de la librería numpy que primero convierte X en un arreglo numérico y luego que divide entre 255 para normalizar los datos.

Una vez normalizado el vector que contiene las imágenes, se convierten las etiquetas en una matriz.

Adicionalmente, en esta función se hace una separación de los datos con la librería sklearn. Esta tiene una instrucción que recibe tres argumentos: las imágenes, las etiquetas y un porcentaje. Este porcentaje indica cuantas de las fotos se van a usar para testeo, es decir, se va a separar los datos de entrenamiento en 2 partes de las cuales una se va a usar para el

entrenamiento y la otra no, esta última se va a usar para comprobar la precisión del modelo clasificando imágenes del dataset.

La función devuelve cuatro elementos, las imágenes de entrenamiento y testeo (x_train y x_test respectivamente) y las etiquetas de entrenamiento y testeo (y_train e y_label respectivamente).

```
# Defino una función para procesar los datos

def preprocess_data(x, y):
    np_X = np.array(x)                                          # Meto los datos en un Array
    normalised_X = np_X.astype('float32')/255.0                 # Normalizo los datos
    label_encoded_Y = utils.to_categorical(y)                   # Los convierto en una matriz

    x_train, x_test, y_train, y_test = train_test_split(normalised_X, label_encoded_Y, test_size = 0.20)   # Separo los datos tal que un 25% son de validación
    return x_train, x_test, y_train, y_test                     # Devuelve los datos en matriz

# Llamo la función y la guardo en variables

x_train, x_test, y_train, y_test = preprocess_data(X, Y)
```

Figura 3.3.2.5 Definición y llamada de la función “preprocess_data”

Con estas dos funciones que se han definido se ha realizado el procesamiento de los datos y están preparados para ser introducidos en la red neuronal.

3.3.3. Entrenamiento y evaluación del modelo

Para el entrenamiento, se definen algunos parámetros:

- **Batch:** Es una agrupación que se hace para optimizar el entrenamiento, en lugar de introducir todos los datos y actualizar los pesos del modelo se evalúan un pequeño lote (batch) y se actualizan los pesos para ese lote, la operación se repite hasta que todos se hayan evaluado todos los datos.

- **Epoch:** Una época (epoch) es una pasada completa a todos los datos de entrenamiento, en esta se han procesado los datos y actualizado los pesos del modelo. El número de épocas es el número de veces que el modelo analiza todos los datos de entrenamiento. Hay que tener en cuenta que un número demasiado grande de épocas puede provocar un sobre aprendizaje del modelo, que se traduce en que ha aprendido a clasificar las imágenes de “memoria” con lo que, aunque tenga una muy buena precisión clasificando fotos del propio dataset, si se introduce una foto que no sea del conjunto de entrenamiento su precisión se reduce mucho.

```
batch = 32
epochs = 50
```

Figura 3.3.3.1 Parámetros de la CNN

En cuanto al optimizador, se ha usado 'adams' que es muy popular por optimizar el aprendizaje de los modelos de forma rápida y eficiente. Y para la función de pérdida se ha usado 'categorical_crossentropy' para medir el error de las predicciones del modelo, esta función es muy usada en modelos multiclase.

Para poder evaluar el rendimiento del modelo durante el entrenamiento se va a evaluar la precisión del modelo, viendo la proporción de muestras correctamente clasificadas.

```
# Defino el optimizador, la función de pérdida y la métrica
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figura 3.3.3.2 Optimizador y función de pérdida de la CNN

Una vez definidos los parámetros necesarios para el entrenamiento, se procede a entrenar el modelo. Para ello la librería keras proporciona la instrucción '.fit' en la cual se introducen los datos de entrenamiento y de testeo, el batch y las épocas antes mencionadas. Además de estos parámetros se añaden dos propiedades:

- **Shuffle:** Esta propiedad puede tener valor 'True' o 'False', y su función es barajar aleatoriamente los datos antes de empezar una época nueva, ayudando a la generalización del modelo.
- **Verbose:** Es una propiedad que permite elegir la información que se puede ver, en este caso se ha elegido '1' que es el modo que más información proporciona.

```
# Entreno La CNN
history = model.fit(x_train, y_train, batch_size=batch, epochs=epochs, validation_data=[x_test, y_test], shuffle = True, verbose=1)

Epoch 1/50
85/85 [==============================] - 6s 19ms/step - loss: 2.9595 - accuracy: 0.2833 - val_loss: 6.1211 - val_accuracy: 0.0489
Epoch 2/50
85/85 [==============================] - 1s 14ms/step - loss: 1.4164 - accuracy: 0.5763 - val_loss: 9.5363 - val_accuracy: 0.0400
Epoch 3/50
85/85 [==============================] - 1s 13ms/step - loss: 0.8845 - accuracy: 0.7230 - val_loss: 9.5752 - val_accuracy: 0.0504
Epoch 4/50
85/85 [==============================] - 1s 14ms/step - loss: 0.6350 - accuracy: 0.7904 - val_loss: 7.0366 - val_accuracy: 0.1644
Epoch 5/50
85/85 [==============================] - 1s 13ms/step - loss: 0.4730 - accuracy: 0.8522 - val_loss: 5.4571 - val_accuracy: 0.1733
Epoch 6/50
85/85 [==============================] - 1s 13ms/step - loss: 0.4164 - accuracy: 0.8670 - val_loss: 3.4858 - val_accuracy: 0.3570
Epoch 7/50
```

Figura 3.3.3.3 Entrenamiento de la CNN

Cuando el modelo ha acabo de entrenar, se hace una comprobación de la CNN en la que se puede observar la precisión final del modelo al clasificar el conjunto de datos de testeo.

```
# Compruebo La CNN

test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)
```

```
22/22 [==============================] - 0s 5ms/step - loss: 0.0352 - accuracy: 0.9881
Test accuracy: 0.9881481528282166
Test loss: 0.035159964114427567
```

Figura 3.3.3.4 Comprobación de la precisión de la CNN

Para finalizar, el modelo entrenado se guarda en una carpeta para poder exportarlo a otros programas.

```
save_options = tf.saved_model.SaveOptions(experimental_io_device="/job:localhost")
model.save('C:\\Users\\santi\\JupyterProyects\\Cap_image\\SCL_model_tf6', options = save_options)
```

Figura 3.3.3.5 Guardado del modelo

3.4. Implementación del programa de clasificación en tiempo real

Este es el programa final, que va a cargar el modelo pre-entrenado y va a tomar una foto de la lo que ve la cámara, esta foto se va a recortar del mismo modo que en el programa de creación del dataset, es decir, se dibuja un cuadro alrededor de la mano y se extrae esa área de interés.

Una vez tiene la foto hecha, la redimensiona para poder introducirla al modelo y hace la predicción. Esta predicción se debe imprimir en el display SSD 1306.

3.4.1. Captura de imágenes con OpenCV y mediapipe

La captura de imágenes se hace del mismo modo que en el programa de creación del conjunto de datos, se usa un bucle que abre la captura de imagen de la cámara. La imagen se observa en una ventana emergente para que el usuario pueda estar al tanto de lo que se va a capturar. El programa espera a que se introduzca una señal por teclado (q) para salir del bucle, guardando la última captura que se ha hecho, deshabilitando la cámara y cerrando la ventana emergente.

En el bucle además de hacer la captura, se usa la librería mediapipe para extraer las coordenadas con las que se va a dibujar el cuadro que marca el área de interés y se va a recortar la imagen.

```
while (capture.isOpened()):

    ret, frame = capture.read()

    color = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)          #Hay que pasarlo a RGB porque mediapipe así lo requiere
    resultado = manos.process(color)                        #Se hace el procesado de la imagen con el que se identifican los 21 puntos
    posiciones = []                                         #En este array vamos a almacenar las coordenadas de los puntos

    if resultado.multi_hand_landmarks:                      #Si se ha detectado algo entra al if
        for mano in resultado.multi_hand_landmarks:         #Busca la mano dentro de la lista de manos que nos da el descriptor
            for id, lm in enumerate(mano.landmark):         #Se obtiene la informacion de cada mano encontrada por el ID
                #print(id,lm)                               #Como entrega decimales (Proporcion de la imagen) hay que pasarlo a pixeles
                alto, ancho, c = frame.shape                #Extraigo el ancho y el alto de los fotpgramas para multiplicarlos por la proporcion
                corx, cory = int(lm.x*ancho), int(lm.y*alto) #Extraigo la ubicacion de cada punto que pertence a la mano en coordenadas
                posiciones.append([id,corx,cory])
#               dibujo.draw_landmarks(frame, mano, clase_manos.HAND_CONNECTIONS)
            if len(posiciones) != 0:
                pto_i5 = posiciones[9]                      #Punto central
                ancho, alto = (pto_i5[1]),(pto_i5[2])       #Obtengo el punto incial del punto central
                x1,y1 = (ancho-75),(alto-75)               #Establezco x1 e y1 en la esquina superior izquierda
                x2,y2 = (ancho+75), (alto+75)              #Establezco x2 e y2 en la esquina inferior derecha
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 3)     #Dibuja un rectangulo en la zona de interés (con centro en el punto antes dicho)

    if (cv2.waitKey(1) == ord('q')):
        break

    #cv2.imwrite(images_path,frame)
    cv2.imshow('webCam',frame)

# Recortar la ROI
roi = frame[y1:y2, x1:x2]                                   #Recorto la imagen para quedarme sólo con la zona de interés antes calculada

# Guardar la ROI como una imagen separada
cv2.imwrite('pred.jpeg', roi)                               #Guardo la imagen recortada con el nombre "pred.jpg"

capture.release()                                           #Dejo de capturar video
cv2.destroyAllWindows()                                     #Cierro la ventana de la cámara
```

Figura 3.4.1.1 Programa que extrae la región de interés y lo guarda en una foto.

Con la foto guardada en una carpeta y con un nombre específico, se procede a cargar esa foto en concreto y su redimensionado para poder introducirla en el modelo. Además del redimensionado el modelo espera una imagen normalizada puesto que así se hizo en el entrenamiento, con lo que la nueva imagen también se ha de normalizar.

```
#Hago la predicción de la imagen

collected_img_path = "C:\\Users\\santi\\JupyterProyects\\mediapipe_test\\pred.jpeg"

collected_img = cv2.imread(collected_img_path, 0)                                   # Leo la imagen guardada (podria usar la variable frame)
collected_img_resized = cv2.resize(collected_img,(75,75))
collected_img_resized = collected_img_resized.reshape((75,75,1))                    # Redimensiono la imagen para meterla al modelo
collected_img_resized = collected_img_resized.astype('float32')/255.0
plt.imshow(collected_img_resized, interpolation='bicubic')                          # Visualizo la imagen guardada
plt.show()

list_image = np.resize(collected_img_resized, (1,75,75,1))
predic_label = model.predict(list_image)                                            # Llamo a la función de predicción
```

Figura 3.4.1.2 Procesado imagen y clasificación

Con el preprocesado hecho, el clasificador intenta predecir qué es lo que se visualiza en la imagen y devuelve en una variable el vector que contiene la probabilidad que tiene cada uno de los elementos. Por supuesto, esto es una forma poco visible de saber qué salida ha dado el clasificador, por lo que se usa la librería numpy para extraer la posición que tiene el número con mayor probabilidad, resultando la posición en el vector que tiene dicha probabilidad. Sabiendo la posición para poder indicar la letra que se ha reconocido, hay que hacer uso del vector que contiene todas las categorías e imprimir el elemento que tenga ducha posición.

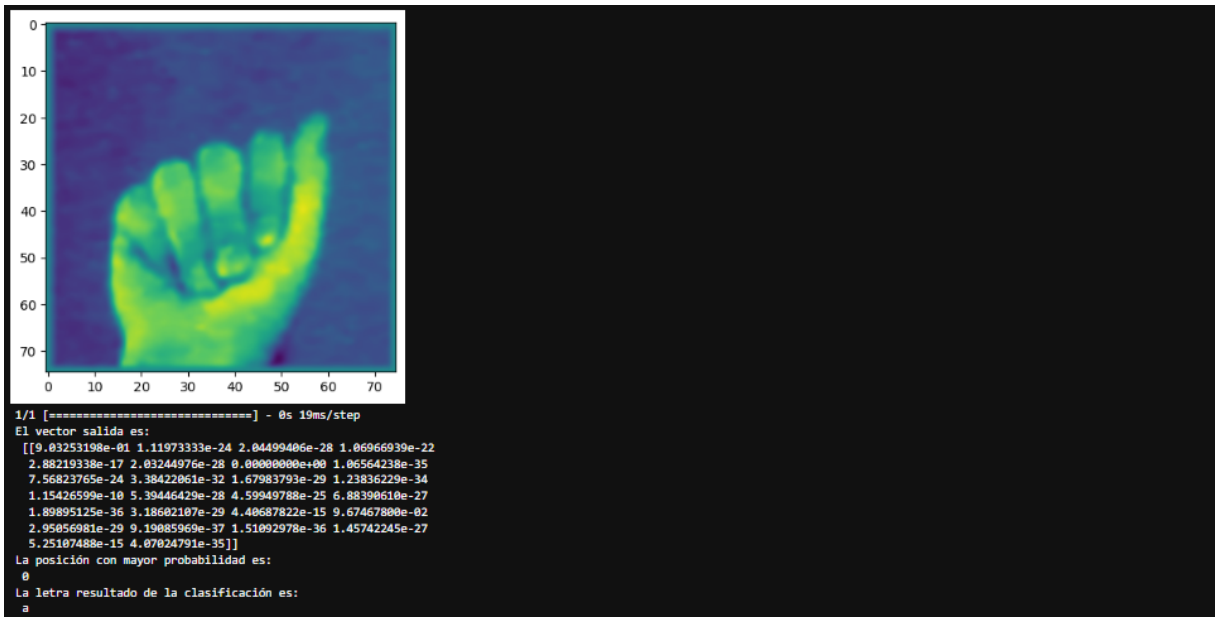


Figura 3.4.1.3 Clasificación de una imagen de prueba

En este punto, se tiene preparado el programa para su uso en un ordenador, pero se tiene que ejecutar en una Raspberry pi 4. Por lo que ahora se va a proceder a la configuración y adaptación del programa para poder ser usado en dicho microordenador.

También hay que tener en cuenta que para el uso del display SSD 1306, hay que usar una librería especial para poder actuar sobre él desde Python. Esta librería es “Adafruit SSD 1306”.

3.4.2. Configuración Raspberry

Para configurar la Raspberry pi 4, se necesitan:

- Microordenador Raspberry pi 4
- Cable Ethernet
- Cable de alimentación
- Tarjeta microSD
- Conexión a internet
- Adaptador microSD a SD normal

Lo primero que se necesita es el sistema operativo que se va a usar, en este caso se ha elegido usar el sistema operativo oficial Raspbian, proporcionado por la propia empresa de Raspberry.

Para descargar Raspbian hay que hacer uso de una herramienta que también la proporciona la empresa de Raspberry. Esta es Raspberry Pi Imager (ver figura ---), que permite formatea la microSD e instala el sistema operativo elegido.

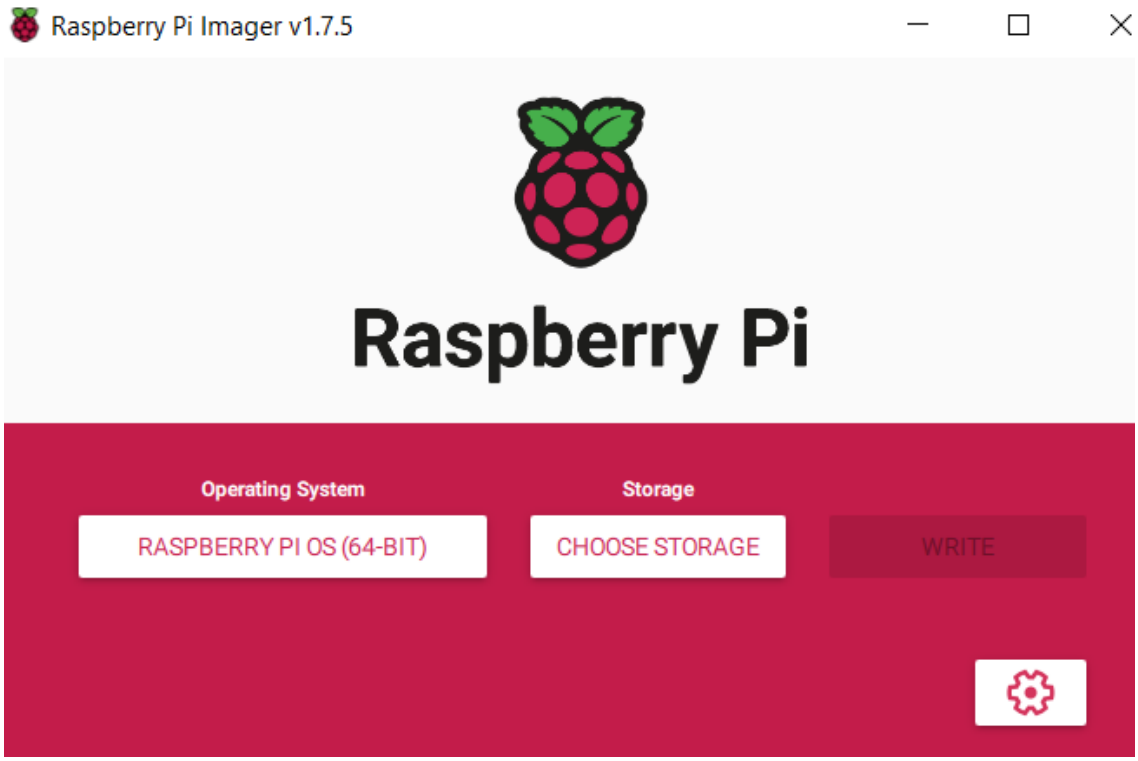


Figura 3.4.2.1 Instalador de Sistema operativo Raspbian

Para esto se conecta la microSD al ordenador, se elige el sistema operativo y la tarjeta de almacenamiento. Para el sistema operativo se ha elegido el de x64 bits, es importante tener en cuenta si se instala el de x32bits que las últimas versiones de TensorFlow no son compatibles por lo que se recomienda seleccionar el de x64 bits. Antes de instalar el sistema operativo, se van a modificar algunas características en los ajustes:

- Se habilita el SSH
- Se establece un usuario y una contraseña

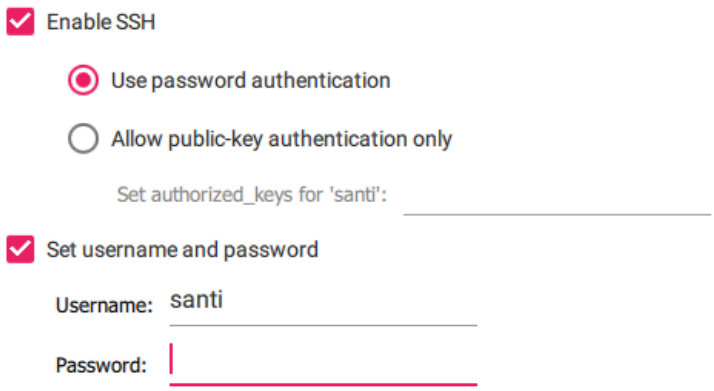


Figura 3.4.2.2 Activación de SSH

Con esto hecho, se puede retirar la SD y conectar la microSD a la Raspberry pi 4. A veces puede dar un fallo en la activación del SSH, para solucionarlo hay que crear un archivo sin extensión en la SD con nombre "SSH" una vez se ha instalado el sistema operativo.

Se va a acceder a la Raspberry en remoto usando el SSH, para ello se necesitan dos aplicaciones extra:

- **Advanced IP scanner:** Esta se usa para obtener la dirección IP de todos los aparatos conectados a la red.
- **RealVNC viewer:** Esta aplicación es un software que permite acceder de forma remota a un ordenador, usando el protocolo VNC (Virtual Network Computing).

Hechos los pasos previos, hay que introducir la tarjeta microSD si no se ha hecho aún, conectar el cable ethernet en el puerto correspondiente y conectar la alimentación del microordenador. Estando conectada a la red, se puede hacer uso de Advanced IP scanner para obtener la IP de la Raspberry.

Para empezar la configuración, desde el ordenador y con la aplicación de símbolo de sistema, se tiene que activar el VNC para poder conectarse en remoto. En el símbolo de sistema se introduce "ssh user@ip_Raspberry -y", sustituyendo "user" por el usuario que se ha establecido en Raspberry Pi Imager y sustituyendo ip_Raspberry por la IP que se ha obtenido con Advanced IP scanner. Ejecutar el comando va a permitir acceder a la configuración después de introducir la contraseña que se definió también en Raspberry Pi Imager.

El propio programa va a mencionar que para acceder a la interfaz de configuración, se debe ejecutar el comando “raspi-config”, el cual hará emerger una ventana para la configuración en la que se va a activar el protocolo VNC en las opciones de interface. También se podría activar en ese mismo menú el bus I2C que usa el display SSD1306, sin embargo, se va a activar más adelante para mostrar cómo se haría desde dentro de la Raspberry.

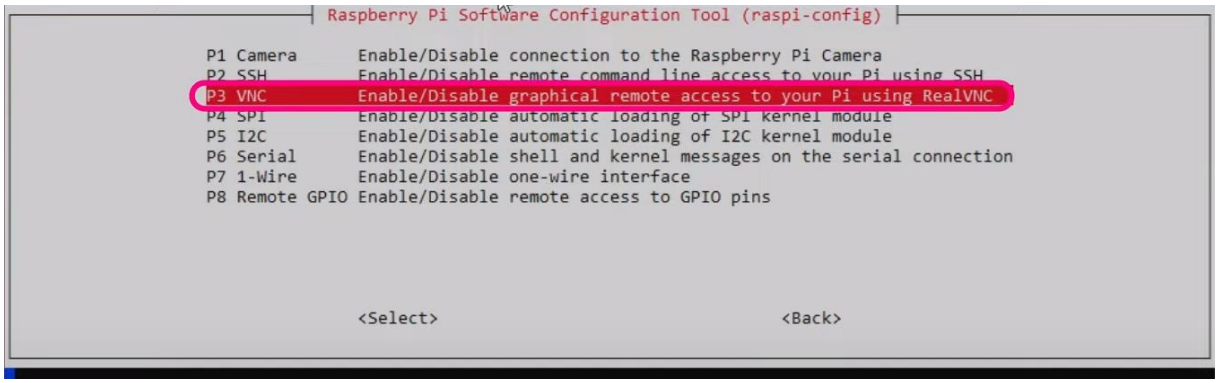


Figura 3.4.2.3 Activación de VNC

Con todo hecho, se procede a usar RealVNC Viewer en el cual se introduce la ip asociada a la Raspberry y se podrá observar la interfaz operativa de Raspbian.

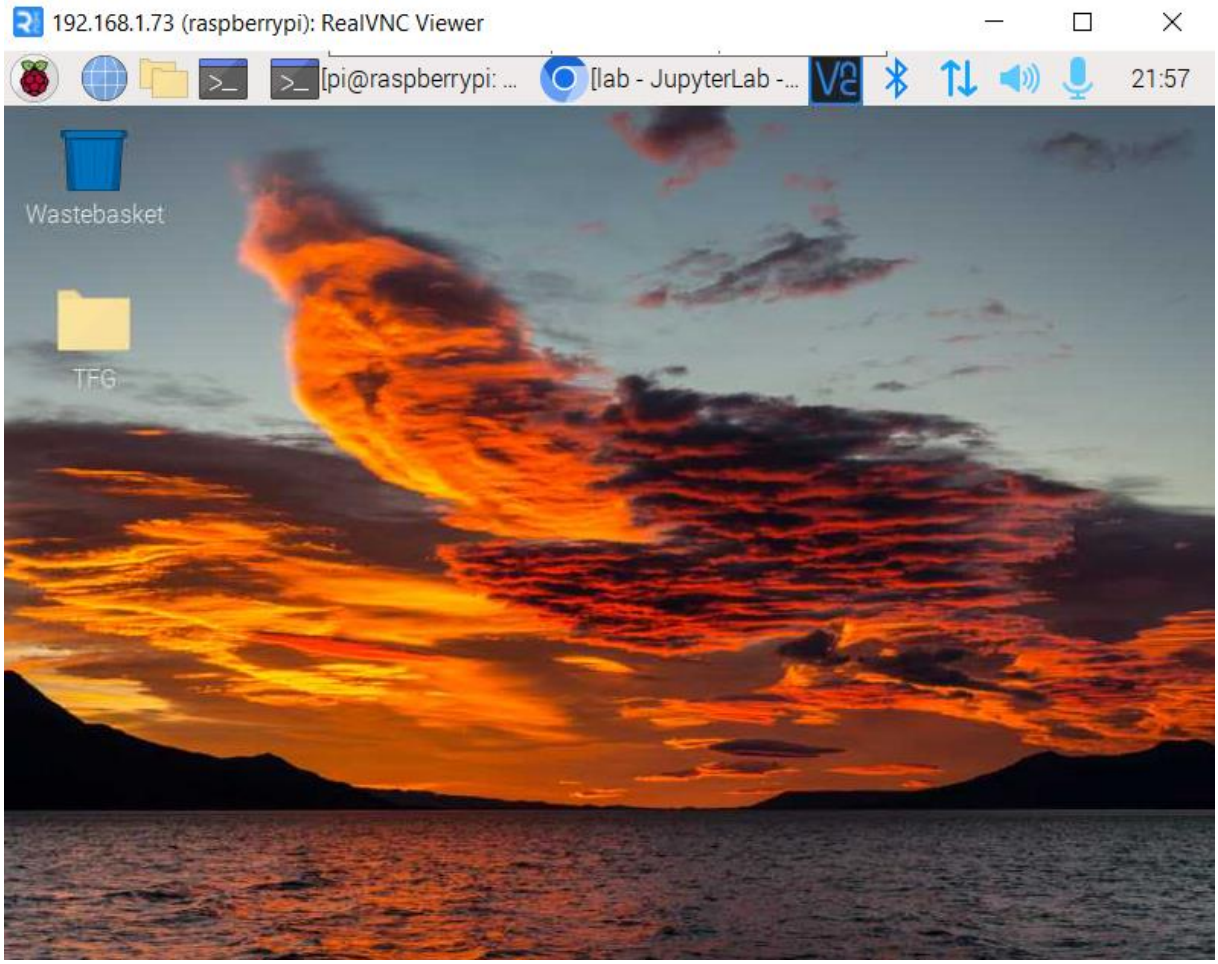


Figura 3.4.2.4 Pantalla de inicio Raspberry

3.4.3. Creación entorno

Si se han hecho correctamente los pasos antes descritos, la Raspberry pi debería estar preparada para su uso por lo que se procede a la creación de un entorno virtual que contenga las librerías necesarias para la ejecución del programa. Las librerías a instalar son:

- OpenCV
- TensorFlow
- Mediapipe
- Adafruit_SSD1306

La primera biblioteca que instalar será TensorFlow, esta tiene un problema general y es que no tiene soporte para Raspberry por lo que para poder instalarla se necesita seguir unos pasos

previos para su descarga con Piwheels. Piwheels es un proyecto que ofrece un repositorio de paquetes ya compilados para la plataforma Raspberry Pi, proporcionando una forma más rápida y conveniente de instalar paquetes de Python.

Para empezar, se van a revisar las características del sistema operativo Raspbian con el comando “cat /etc/os-release”. Se puede observar que la instalación elegida (Raspbian x64 bits) está desarrollado con Linux 11 (Bullseye).

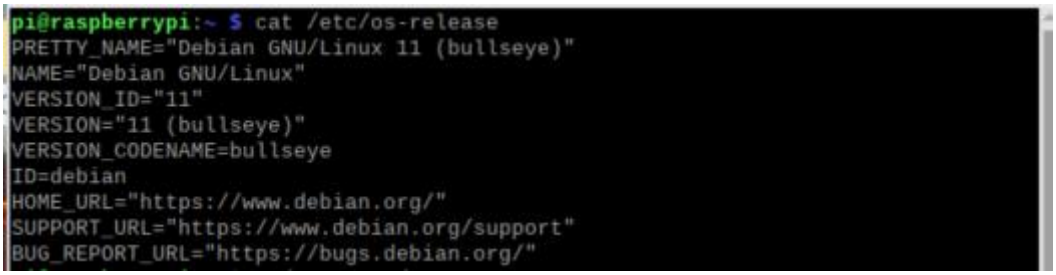


Figura 3.4.3.1 Características del SO Raspbian

Es recomendable que, antes de hacer cualquier instalación se actualice el sistema de la Raspberry Pi son los comandos “sudo apt update” y “sudo apt upgrade -y”.

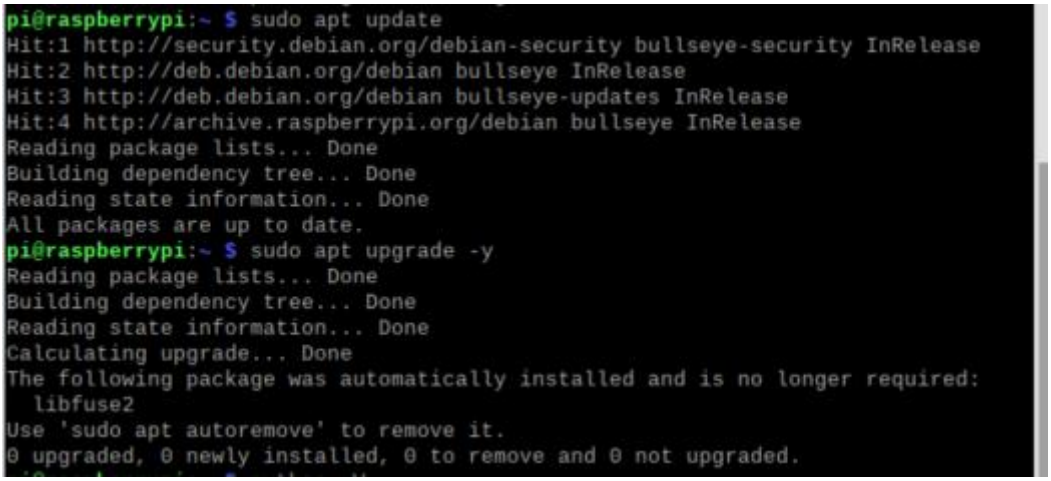


Figura 3.4.3.2 Actualizar sistema Raspberry

Con todo actualizado se procede a ver la versión de Python y la arquitectura que se está utilizando, esto es importante para saber qué versión de TensorFlow se tiene que instalar. Como se puede ver en la figura ----, en este proyecto la Raspberry tiene Python 3.9.2 y una arquitectura ARM64.

Figura 3.4.3.3 Versión de Python y arquitectura (arm architecture)

Para poder crear un espacio virtual es necesario instalar los paquetes “Virtualenv” y “Python3”, que se utilizan con sistemas basados en Debian para crear entornos aislados para desarrollar y ejecutar programas de Python con sus propias dependencias y configuraciones.

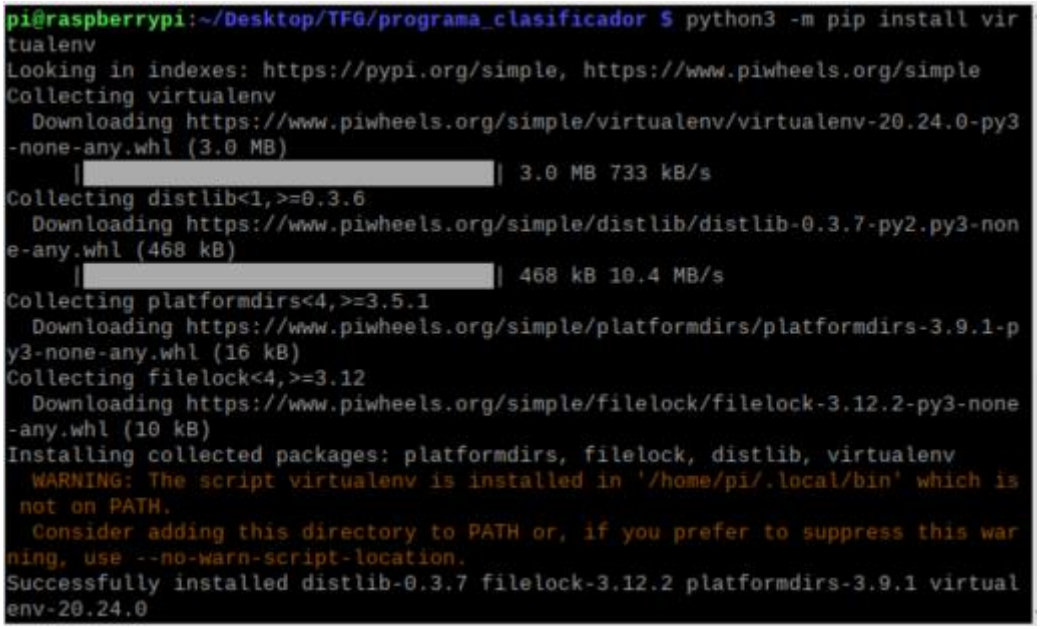


Figura 3.4.3.4 Descarga de creador de entornos

Se procede a la creación y activación de un entorno en el que se van a instalar las librerías necesarias para la ejecución del programa, a este entorno se le va a llamar “tf” y se va a guardar en una carpeta especificada destinada a contener todo el contenido del programa objeto de este proyecto.

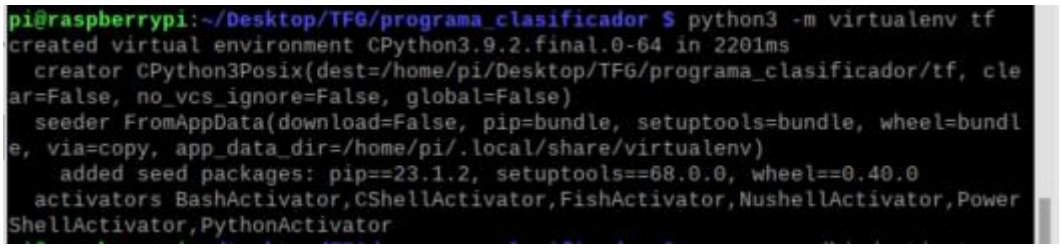


Figura 3.4.3.5 Creación de un entorno virtual tf

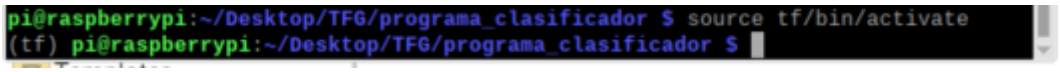


Figura 3.4.3.6 Activación del entorno

Como se ha mencionado antes, TensorFlow no tiene soporte para Raspberry por lo que hay que instalar de forma manual las dependencias necesarias.

Figura 3.4.3.7 Instalación de dependencias

Una vez se instalan las dependencias, se debe descargar el instalador de TensorFlow, en este momento es donde hay que tener en cuenta si se ha instalado Raspbian x64 bits o x32 bits y que la arquitectura es ARM, ya que existen limitaciones para la versión x32 bits. Si se hace uso de armv7 (x32 bits) no se tiene compatibilidad con Python 3.8 o mayor, en ese caso o bien se tiene que bajar la versión de Python o bien se ha de reinstalar el sistema operativo con la versión de x64 Bit. A continuación, se expone un par de ejemplos de las combinaciones de compatibilidad:

Si se tiene Python versión 3.9.* (“ * ” significa cualquier número) y arquitectura “aarch64”, se puede instalar la versión 2.9.0 de TensorFlow. Esta versión es compatible con la arquitectura ARM de x64 Bits.

Si se tiene Python versión 3.7.* (nótese que es <3.8.*) y arquitectura “armv7”, se puede instalar la versión de TensorFlow 2.5.0 o inferior, las versiones más actualizadas no serían compatibles.

Si se tiene Python versión 3.9 y arquitectura “armv7” se tiene que instalar una versión inferior de Python o reinstalar el sistema operativo Raspbian de x64 Bits.

Para este proyecto se ha instalado la versión 2.9.0 de TensorFlow con Python 3.9.2 y que tiene soporte a la arquitectura ARM.

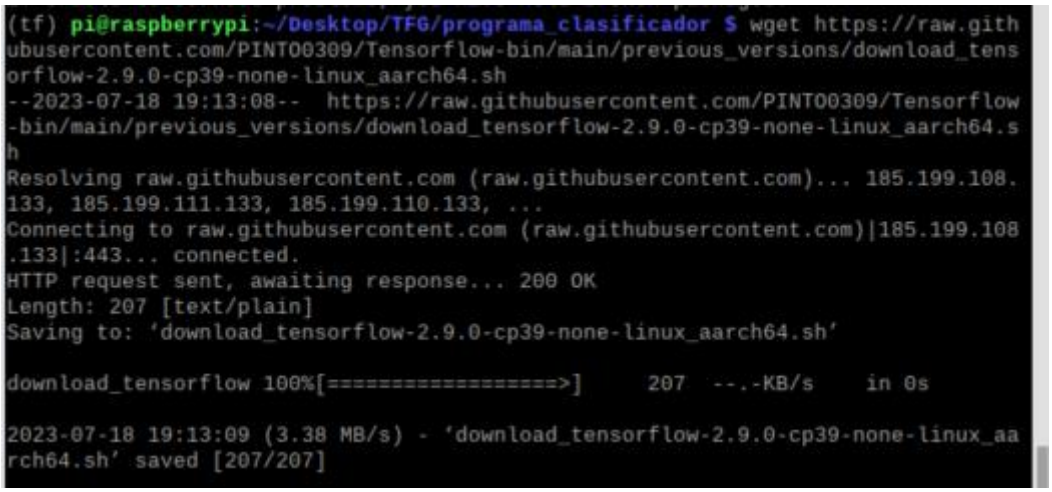


Figura 3.4.3.8 Descarga de una versión de TensorFlow desde GitHub

Solo falta instalar el archivo ".whl" (Wheel), esta viene dentro del archivo de TensorFlow que se descargó en el paso anterior, así que se descarga del mismo como se ve en la figura 3.4.3.9.

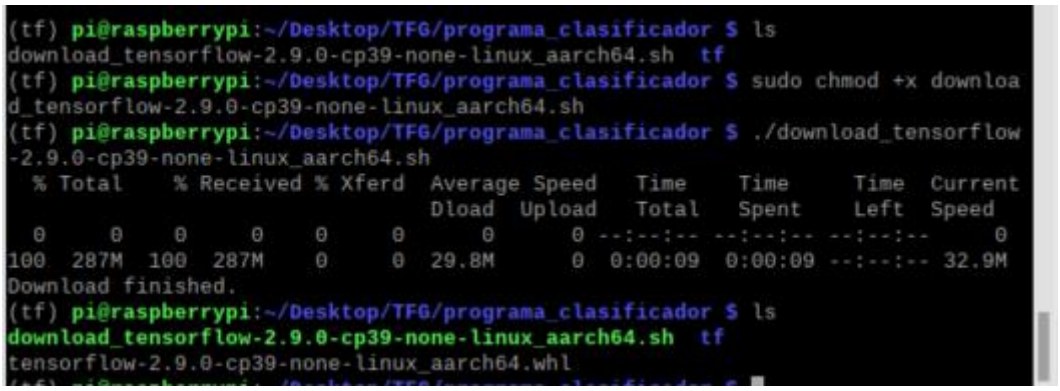


Figura 3.4.3.9 Creo el archivo Wheel que se usara para instalar TensorFlow

Con todo preparado, se procede desinstalar versiones de TensorFlow si las hubiese para evitar compatibilidad y una vez comprobado, se instala TensorFlow usando "pip install" y el archivo ".whl" que se descargó antes. Con esto inicia la descarga que tarda unos minutos dependiendo de la potencia de la Raspberry y el sistema operativo.

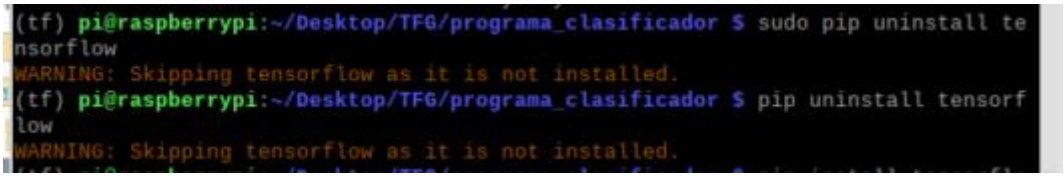


Figura 3.4.3.5 Comprobación de que no haya versiones antiguas de TensorFlow

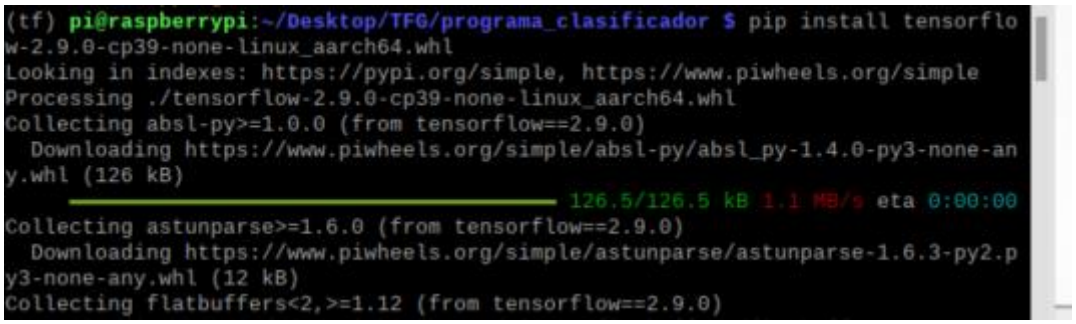


Figura 3.4.3.6 Instalación de TensorFlow con el archivo Wheel que se descargó antes

Ahora se va a instalar la librería de OpenCV, esta no tiene problema para su instalación por lo que simplemente se hace uso de “pip install OpenCV-py” para instalarla.

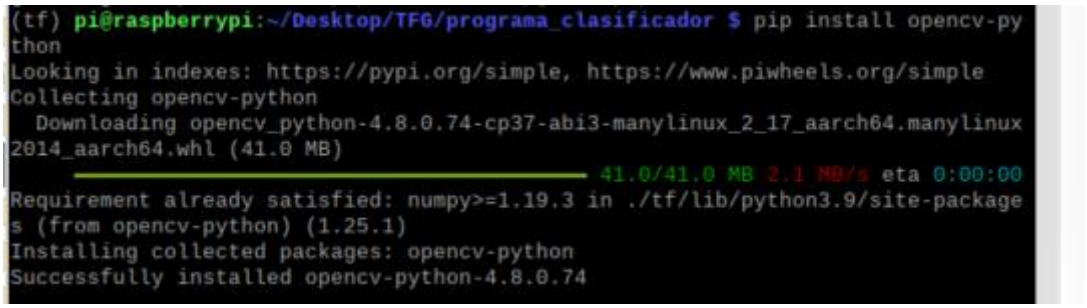


Figura 3.4.3.7 Instalación OpenCV

Se hace una comprobación de que ambas bibliotecas están bien instaladas abriendo Python, importando las librerías y ejecutando el comando “librería.__version__” donde librería es la instancia con la que se ha importado. Como se puede ver en la figura 3.4.3.7, se han instalado correctamente.

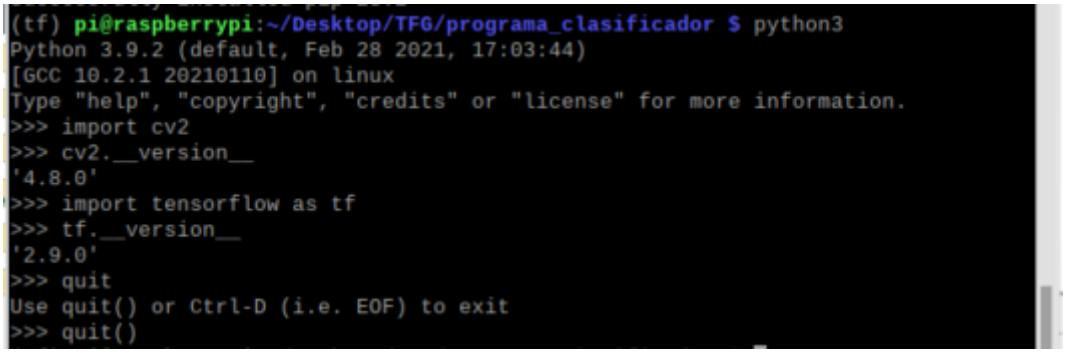


Figura 3.4.3.8 Comprobación de que están instalados

Para hacer una segunda comprobación del entorno que se ha creado, se ejecuta un pequeño programa que simplemente muestre la pantalla usando el editor de código Thonny. Este editor de código viene instalado por defecto en la Raspberry. Hay que tener en cuenta que para usar el entorno virtual creado hay que acceder al menú de la esquina inferior derecha, abrir el configurador del interprete (análogo al Kernel de Jupyter Lab) y elegir la ruta del entorno antes creado.

Figura 3.4.3.9 Con el editor de código Thonny, usar el intérprete para abrir el entorno con las librerías antes añadidas

```
import cv2

cap = cv2.VideoCapture(0)

while True:

        ret, frame = cap.read()

        cv2.imshow('Video', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```

Figura 3.4.3.10 Programa de comprobación

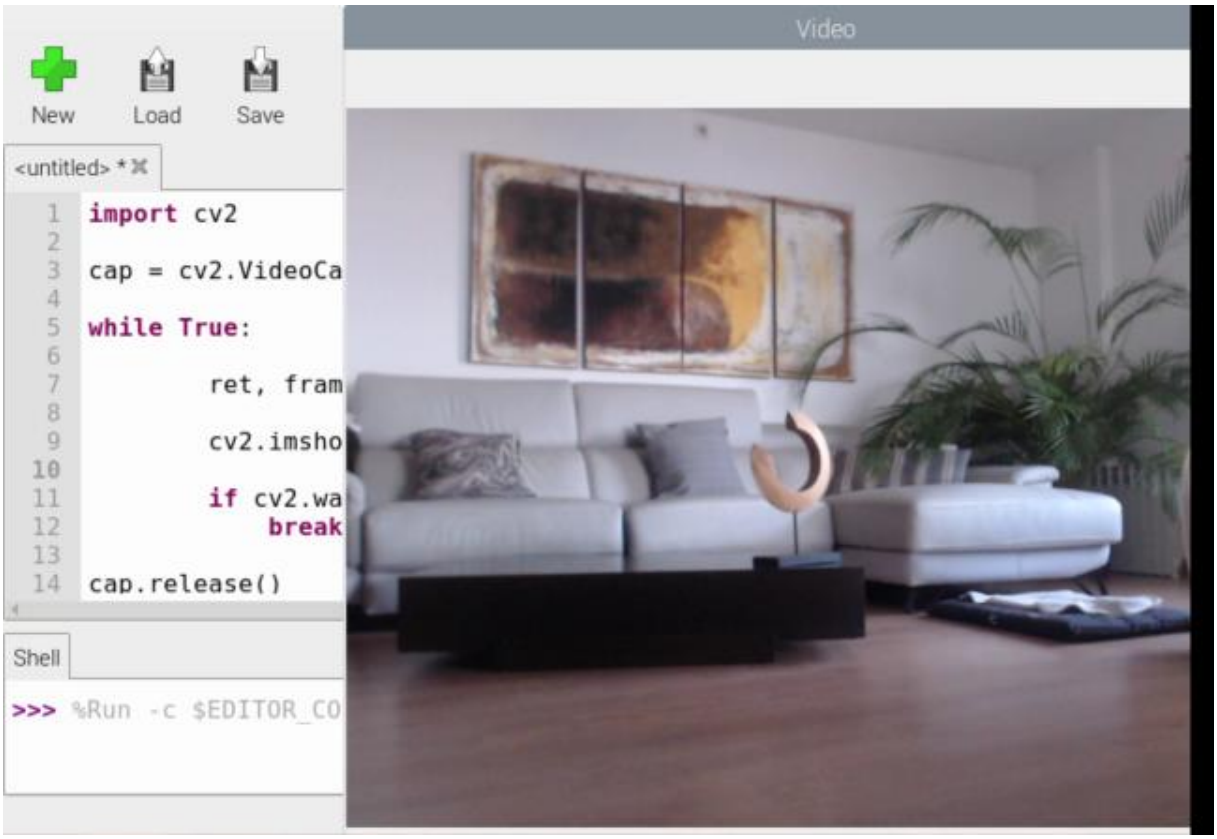


Figura 3.4.3.11 Resultado correcto del programa de prueba

Con las estas dos librerías disponibles y funcionando correctamente se procede a instalar mediapipe, esta librería tampoco tiene complicación y es suficiente con usar el comando "pip install mediapipe". Esta librería instala también una versión más antigua de OpenCV por defecto, pero se ha considerado más conveniente hacer una instalación manual para asegurar su correcto funcionamiento.

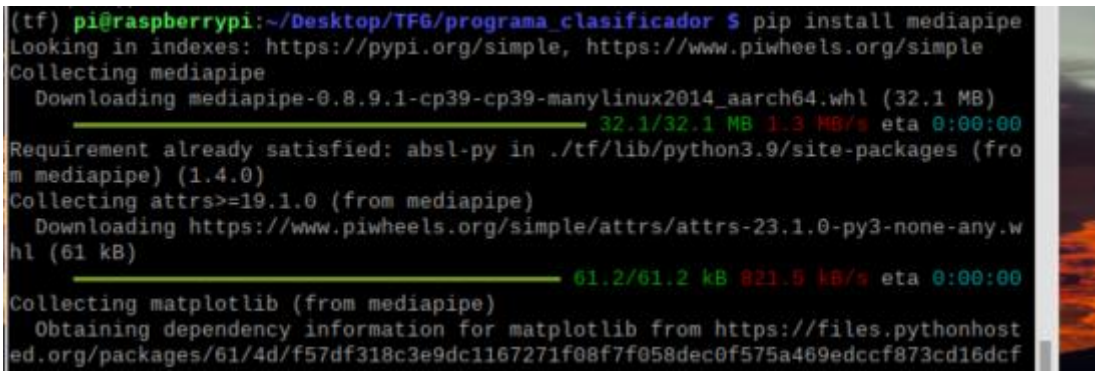


Figura 3.4.3.12 Instalación de Mediapipe

Para termina con el entorno virtual, es necesario instalar la librería que se encarga de gestionar la pantalla OLED SSD1306 con Python, esta es adafruit. Esta librería es de libre uso y para su instalación se puede acceder a GitHub donde se pueden obtener varias versiones de esta. Siguiendo las instrucciones que indica el desarrollador, para la instalación de la librería y otros paquetes necesarios se usa el comando “pip3 install adafruit-circuitpython-ssd1306”

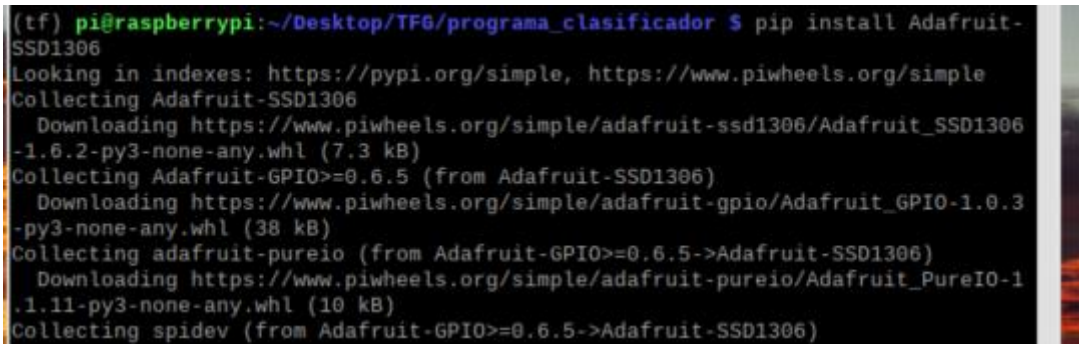


Figura 3.4.3.13 Instalación de la librería que gestiona el SSD1306 con Python

Una vez completada la configuración previa del entorno virtual que se va a usar, se va a instalar un editor de código específico ya que es con el que se han creado todos los programas. Este editor de código es Jupyter Lab y se ha elegido este para la creación de todos los programas por lo intuitivo que es, además tiene una característica que es de mucha ayuda y es la posibilidad de ejecutar solo trozos del código ya que en Jupyter Lab se pueden añadir celdas y cada una puede ser ejecutada de forma independiente haciendo uso de las variables que se han usado y modificado en las celdas anteriores, esta característica es especialmente útil en el programa de entrenamiento.

Para instalar Jupyter Lab, simplemente se usa el comando “pip install jupyterlab”. Una vez instalado para abrirlo estando en el entorno virtual hay que navegar en la consola usando el comando “cd” para entrar en la carpeta donde está el entorno, ahí se activa el entorno virtual y ya activado se puede abrir Jupyter Lab.

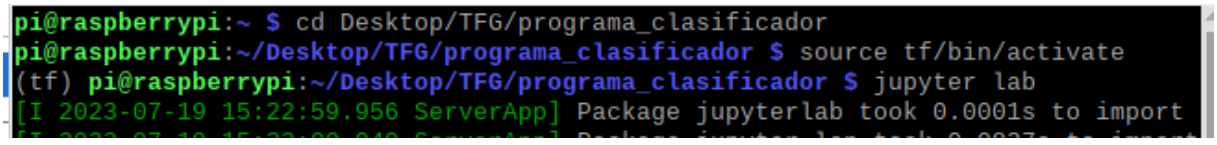


Figura 3.4.3.14 Abrir Notebook de Jupyter Lab

En este punto se tiene ya todo preparado para el uso del programa, solo falta pasar el programa que hace la foto y la pasa por el modelo, y el modelo preentrenado, a la Raspberry. Esto se puede hacer o bien con un elemento de almacenamiento USB que contenga una copia del programa o bien si se quiere enviar por ssh usando el comando "scp archivo origen usuario@direccion_ip_Raspberry:directorio_destino" donde:

Archivo origen: es la ruta completa del archivo que se va a enviar.

Usuario: se introduce el usuario que se estableció en la Raspberry Pi cuando se instaló el sistema operativo (ver figura 3.4.2.1)

Dirección IP Raspberry: la dirección asociada a la Raspberry que se puede conseguir haciendo uso de Advanced IP Scanner como se indicó antes.

Directorio destino: Es la carpera donde se va a guardar el programa.

Además, tener en cuenta que, si se va a enviar una carpeta en lugar de un archivo, antes de origen de usuario hay que añadir "-r ", que copia todos los archivos en la dirección especificada.

SCP significa "secure copy", como su nombre indica hace una copia de los elementos de la dirección origen a la dirección destino.

Realizados todos los pasos correctamente, se tiene el programa disponible para su uso con el editor de código Jupyter Lab. Sin embargo, el programa se limita a hacer la predicción y guardar dicha clasificación en una variable llamada "classes", así que se va a explicar las modificaciones que se tienen que hacer para poder representar dicha letra en el SSD 1306.

3.4.4. Preprocesamiento de imágenes en el display de la Raspberry Pi

Antes de proceder a explicar el programa, se va a explicar el cableado del oled SSD 1306. Para ello se puede acceder al datasheet del fabricante, en el cual se puede observar que la tensión de alimentación puede ser de 3.3V. En la Raspberry Pi 4 se tiene un pin que permite esa tensión (ver figura 2.5.2.4).

También viendo el esquema del GPIO (ver figura 2.5.2.4) se puede observar que se tiene un pin para SDA, SCL y tierra, con lo cual se procede a hacer la conexión (ver figura 3.4.4.1)

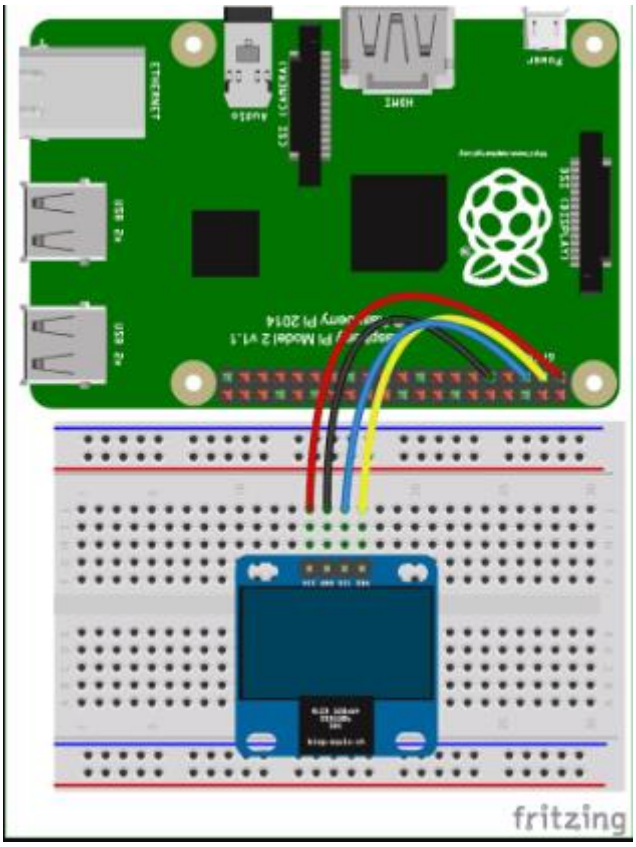


Figura 3.4.4.1 Esquema conexión SSD 1306 a Raspberry Pi 4

Teniendo ya el SSD1306 conectado, se va a explicar el programa. Para el correcto funcionamiento del programa hay que recordar actualizar todas las rutas que se referencian en el programa, actualizando a la ruta asignada en la Raspberry. Además, en la anterior versión del programa para obtener el vector con las categorías a clasificar se hacía con “os.listdir” y se leían las carpetas que existían en la ruta de entrenamiento. Esto se va a modificar por dos razones:

- En la Raspberry sólo está el programa que clasifica la imagen, no contiene la carpeta con los datos de entrenamiento.
- Se ha considerado más conveniente simplemente crearlo manualmente debido a las limitaciones de la Raspberry, esto se hace porque la Raspberry que se ha usado tiene únicamente 2GB de RAM por lo que se quiere reducir todo lo posible el uso de operaciones para evitar problemas.

Por lo tanto, la primera modificación que se ha hecho es crear el vector con las categorías "classes" manualmente (ver figura 3.4.4.2).

```
classes = ['a', 'b', 'c','d', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
print(len(classes))
print(classes)
```

Figura 3.4.4.2 Creación manual del vector "classes"

Con esto y recapitulando un poco, se tiene un programa que carga un modelo, captura una imagen que contiene el área de interés haciendo uso de las librería OpenCV y Mediapipe y se hace pasar la imagen capturada por el clasificador con la librería TensorFlow, una vez hecha la clasificación se guarda en la variable "predic_label" como un vector de probabilidades, con ayuda de numpy usando la instrucción "argmax" se obtiene la posición con la mayor probabilidad, dicha posición es la del vector "classes".

```
El vector salida es:
 [[0.0000000e+00 3.2451097e-33 0.0000000e+00 6.9798486e-38 3.2971763e-21
  2.3327514e-25 2.0721518e-16 5.4521139e-29 9.4472404e-17 0.0000000e+00
  0.0000000e+00 1.3623955e-33 2.8970499e-17 9.9556848e-33 9.2277360e-30
  0.0000000e+00 0.0000000e+00 1.8510303e-36 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 3.8317332e-26 0.0000000e+00 0.0000000e+00
  1.0000000e+00]]
La posición con mayor probabilidad es:
 25
La letra resultado de la clasificación es:
 z
```

Figura 3.4.4.3 Visualización de resultados

Se procede a ampliar el programa para añadir la funcionalidad de pintar la letra resultado de la clasificación en el SSD 1306. Para ello se tienen que importar cuatro librerías que se descargaron antes, estas son:

- **Board:** Proporciona constantes que facilitan el acceso a los pines y características de la Raspberry Pi, por ejemplo, en este caso se usa para poder referenciar los pines GPIO 2 y GPIO 3 como SDA y SCL respectivamente.
- **Busio:** Se usa para crear y configurar interfaces de comunicación, es la que va a hacer posible el uso del bus I2C que usa el SSD 1306.

- **PIL (Python Imaging Library):** Permite la creación y manipulación de imágenes antes de mostrarlas en el SSD 1306.

- **Adafruit_ssd1306:** Como se mencionó antes, es la que permite el uso del SSD 1306.

```
from board import SCL, SDA
import busio
import adafruit_ssd1306
from PIL import Image, ImageDraw, ImageFont
```

Figura 3.4.4.4 Librerías para la gestión del SSD 1306

Con las librerías necesarias instaladas se hace la inicialización de la pantalla y de lo pines.

```
# Inicialización y configuración de OLED SSD1306
i2c = busio.I2C(SCL, SDA)
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
```

Figura 3.4.4.5 Inicialización de SSD 1306

En este momento, finalmente se puede escribir el código que interactúa con el SSD 1306. Para este proyecto se ha optado por la creación de una función que será la encargada de pintar la letra resultado de la clasificación. Esta función se va a llamar "pintar_letra" y espera un único parámetro, que es la letra mencionada (como carácter, no su posición ni el vector).

Función "Pintar_letra"

La función empieza limpiando la pantalla para evitar que se solapen dibujos anteriores con el que se pretende pintar nuevo. Para ello se pinta toda la pantalla de negro.

```
def pintar_letra(letra):
    # Limpio la pantalla
    oled.fill(0)
    oled.show()
```

Figura 3.4.4.6 Limpiar pantalla

Hecho esto, se procede a crear una imagen vacía donde se va a representar la letra, para ello se hace uso de la librería PIL. Esta librería crea una imagen que se va a poder pintar en blanco y negro, ya que se introduce 1 en el primer parámetro, y se establece el tamaño que tendrá, en este caso tendrá el tamaño del SSD 1306. Con la imagen creada, se va a crear un objeto

(draw) asociado a ella, permitiendo dibujar sobre la imagen vacía que se ha creado antes y así agregar contenido visual.

```
# Creo un fondo blanco y un objeto
img = Image.new("1", (oled.width, oled.height))
draw = ImageDraw.Draw(img)
```

Figura 3.4.4.7 Creación de imagen en blanco y objeto para interactuar

Para dibujar la letra, se ha descargado una fuente desde "Google fonts" debido a que la fuente se tiene por defecto es muy pequeña y tiene un tamaño fijo, en cambio las fuentes descargadas tienen mayor variedad de estilos y tamaños.

En este caso se ha descargado la fuente DM Sanz, que tiene un tamaño de hasta 36 pts, en el programa para poder trabajar con esta fuente, se ha añadido en una variable (Font) donde se le ha asignado un tamaño de 24 pts usando la librería PIL.

```
font = ImageFont.truetype("/home/pi/Desktop/TFG/programa_clasificador/Fuentes/DMSans-Italic-VariableFont_opsz,wght.ttf",24)
```

Figura 3.4.4.8 Importar fuente descargada

Para terminar la función se tiene que poner el puntero en la posición donde se va a representar en la pantalla. Puesto que es una única letra se ha decidido pintar dicha letra en el centro de la pantalla. Para ello se han obtenido las coordenadas del cuadro que delimita la letra, con ellas se calcula su ancho y alto. Para las coordenadas se resta el ancho/alto de la letra al ancho/alto de la pantalla y se divide a la mitad, quedando el puntero en una posición que centre la letra.

```
coord = draw.textbbox((0,0),letra, font=font1)
ancho = coord[2] - coord[0]
alto = coord[3] - coord[2]

x = (oled.width - ancho) / 2
y = (oled.height - alto) /2
```

Figura 3.4.4.9 Establecer coordenadas de la letra

Para finalizar la función, simplemente se pinta en la elemento que se creo antes, en este caso se escribe en la posición central indicada por las coordenadas que se acaban de obtener, la letra que se introduce en la función y con la fuente seleccionada, además, al añadir "fill=255"

se indica que lo que se va a pintar va a ser en blanco en contraste al fondo negro. Para hacer visible el dibujo se indica al oled que la imagen que se va a representar es “img” y se actualiza la pantalla.

```
    draw.text((x,y), letra, font=font1, fill=255)
    oled.image(img)
    oled.show()
```

Figura 3.4.4.10 Pintar en la pantalla

Solo queda llamar a la función y se obtiene en el SSD 1306 la letra, dando así por finalizado el paso final del proyecto.

```
letra = classes[np.argmax(predic_label)]
pintar_letra(letra)
```

Figura 3.4.4.11 Llamada a la función

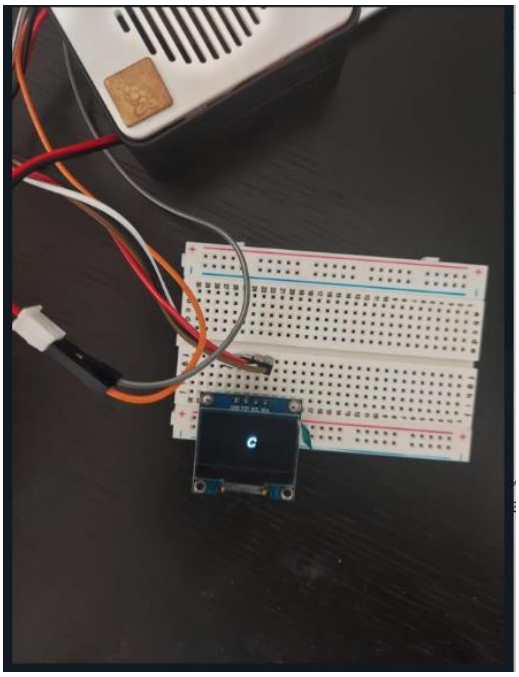


Figura 3.4.4.12 Resultado final

4. RESULTADOS Y EVALUACIÓN

4.1. Evaluación del modelo de clasificación

A continuación, se va a estudiar con dos gráficos el comportamiento de la CNN.

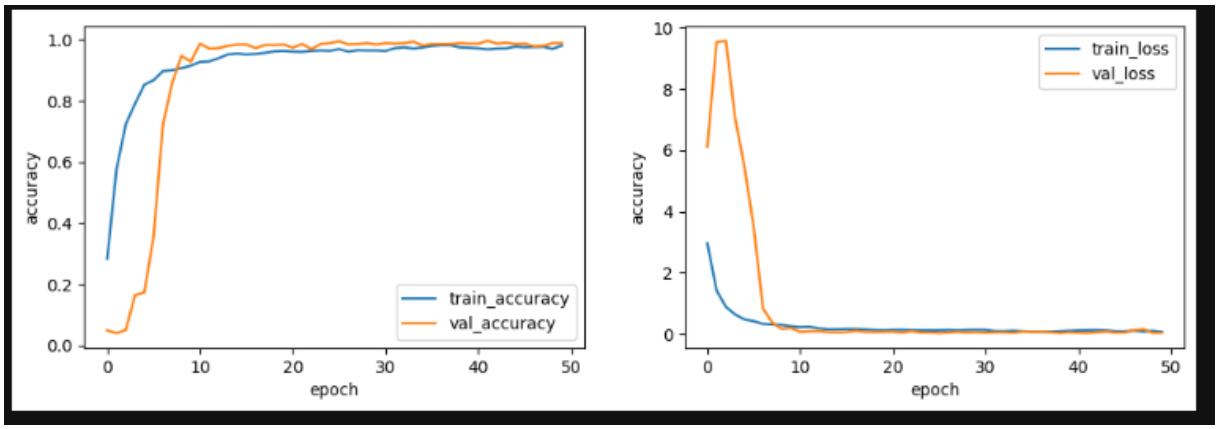


Figura 4.1.1 Gráficas precisión frente épocas y pérdida frente épocas respectivamente

En la primera gráfica se puede ver cómo ha ido evolucionando la precisión frente las épocas y en la segunda se puede ver cómo ha evolucionado la función de pérdida frente a las épocas. En ambas se puede ver que, aproximadamente, en la época 10 el modelo ya casi ha alcanzado una precisión del 97% y una pérdida cercana a 0. A partir de esta época mejora poco sus características, lo que nos indica que no es necesario poner tantas épocas para este modelo.

Por otra parte, para poder ver desde otra perspectiva la precisión, se ha hecho uso de una matriz de confusión y un mapa de calor que permitan ver en una matriz y con colores la concentración de datos, en la diagonal de la matriz se encuentran las predicciones correctas y todo valor que esté fuera de la diagonal son predicciones incorrectas.

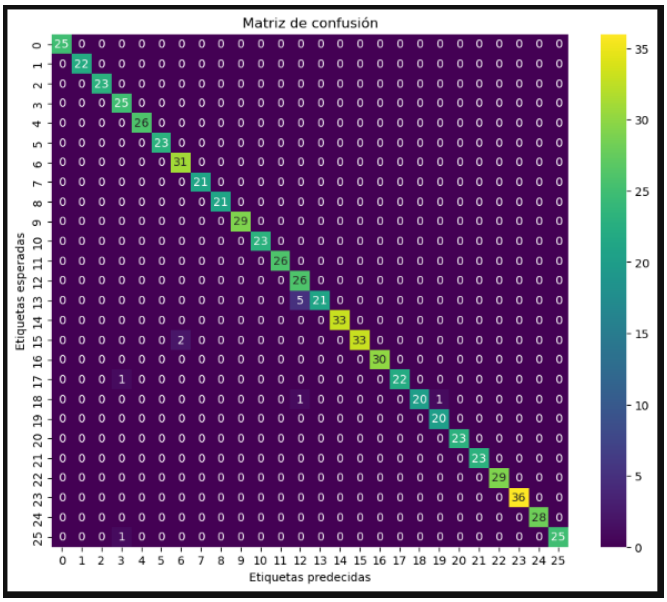


Figura 4.1.2 Matriz de confusión del modelo

Como se puede observar, el modelo ha tenido un muy buen rendimiento a la hora de aprender y clasificar imágenes que forman parte del dataset, pero esto no asegura su correcto funcionamiento con imágenes que estén fuera del mismo.

Como se ha podido observar, en el entrenamiento el modelo ha dado un muy buen comportamiento (ver figura 3.3.3.4), estando por encima del 98%. Sin embargo, se han hecho pruebas con todas las letras usando imágenes hechas con el programa clasificador y se ha podido observar que hay seis letras que no las clasifica adecuadamente, estas son la A, la E, la J, la R, la U y la V. Por ejemplo, la A se clasifica como un S, la E se clasifica como una E.

En cuanto al resto de letras la mayoría la clasifica como se espera, dejando de lado que en algunas ocasiones se equivoca, pero al jugar con la distancia se clasifica correctamente.

En general, puesto que de veintiséis letras solamente seis dan fallo, se puede considerar aceptable teniendo en cuenta las limitaciones que existen.

4.2. Análisis de los resultados obtenidos

Viendo los resultados obtenidos, se puede concluir que la mayoría de los problemas se deben principalmente a dos factores:

- Dataset muy pequeño (130 imágenes).
- Hay signos que son muy parecidos.

Para el caso de signos muy parecidos, como puede ser el caso de la A y la S (ver figura 4.2.1), se puede observar que la única diferencia es la posición del pulgar, haciendo que sea más fácil confundirse.

Figura 4.2.1 Similitud entre signo de la A y la S

4.3. Posibles soluciones

Para el caso de el tamaño del dataset la solución es clara, aumentar el tamaño del dataset. Para ello se tiene que aumentar el número de fotos que se hace en el programa de creación del dataset. Además, sería conveniente aumentar la variedad de las fotos ya sea introduciendo cambios de luz, variar la distancia, introducir manos con características distintas como puede ser en tamaño, color de piel, manos de hombre, mujer, niño, persona mayor, etc. En definitiva, aumentar todo lo que se pueda la variedad y tamaño del dataset.

También se puede considerar el uso de un generador de imágenes en el entrenamiento, este añade distintas variaciones artificiales al dataset (ver figura 4.3.1). En la realización de este proyecto se hizo un intento de modelo usando un generador de imágenes en el cual se añadían imágenes que tenían acercamiento y rotación aleatorio, pero dio un peor funcionamiento.

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=0,
    width_shift_range=0.0,
    height_shift_range=0.0,
    brightness_range=None,
    shear_range=0.0,
    zoom_range=0.0,
    channel_shift_range=0.0,
    fill_mode='nearest',
    cval=0.0,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    interpolation_order=1,
    dtype=None
)
```

Figura 4.3.1 Propiedades del generador de imágenes.

En el caso del acercamiento el fallo puede estar en que al extraer el área de interés con mediapipe y OpenCV, si se acerca la foto, se pierde información como se muestra en la siguiente figura.

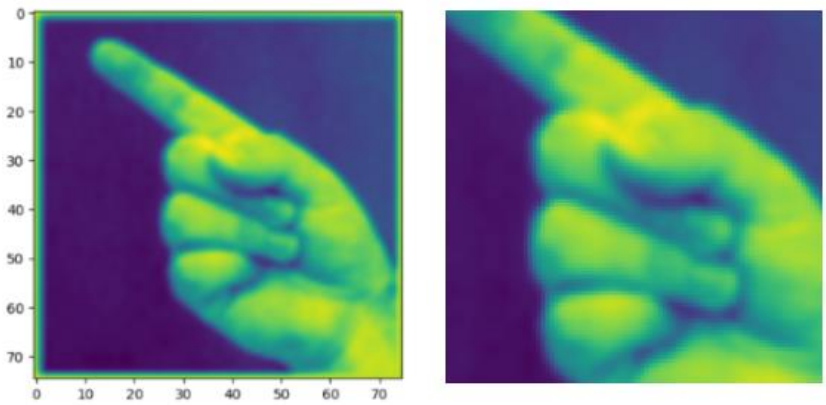


Figura 4.3.2 Pérdida de información debido a acercamiento.

En cuanto a la rotación, da problema debido a que hay letras en que la única diferencia que hay es la inclinación, como es el caso de la i y la j (ver figura 4.3.2) con lo cual, añadir rotaciones hace que el modelo pierda la referencia y no pueda clasificarlas apropiadamente.

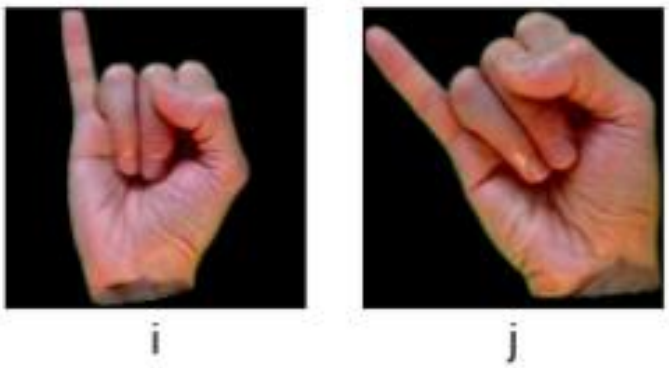


Figura 4.3.3 Similitud entre i y j que impide las rotaciones.

No se añadieron más modificaciones porque se necesitaría un estudio más exhaustivo de qué hace cada modificación y probar su efecto en el dataset.

Otra posible solución ante la similitud de las fotos puede ser variar el tamaño de batch, esto haría que varíe la generalización que se hace en el entrenamiento. El problema de esta solución es que la Raspberry Pi que se ha usado en este proyecto es de 2GB de RAM, con lo cual un aumento del tamaño de batch haría que se consuma más memoria superando las limitaciones. De hecho, con el modelo actual ya se presenta un problema (ver figura 4.3.4). Esta limitación en el hardware hace que haya posibles soluciones difícilmente aplicables.

```
2023-07-19 12:36:42.623509: W tensorflow/core/framework/cpu_allocator_imp
l.cc:82] Allocation of 84934656 exceeds 10% of free system memory.
```

Figura 4.3.4 Aviso de que se está usando demasiada memoria RAM

También se puede probar con distintos tipos de modelo, variando el número de capas que tiene, el índice de abandono, etc. En este caso se ha utilizado la configuración que mejores resultados dio, pero eso no significa que no exista una mejor.

5. CONCLUSIONES

5.1. Resumen de los logros del TFG

En el presente trabajo de fin de grado, se estableció como objetivo desarrollar y evaluar un sistema de clasificación de imágenes utilizando redes neuronales convolucionales y aplicar el sistema clasificador desde un microordenador Raspberry Pi. Para alcanzar estos objetivos, se llevaron a cabo distintas etapas:

- **Investigación:** Se realizo una exhaustiva investigación acerca de redes convolucionales, clasificadores de imágenes y demás técnicas. Además, se hizo una investigación sobre el estado y uso actual del microordenador Raspberry Pi 4.
- **Diseño e implementación del modelo:** Se diseñó una arquitectura de red neuronal convolucional personalizada. Para ello se hizo uso de bibliotecas enfocadas al aprendizaje profundo como TensorFlow y Keras. Además se ha conseguido que dicho modelo clasifique una captura hecha en el momento, de forma que la imagen no es parte del dataset.
- **Adquisición y preprocesado de datos:** Se obtuvo un conjunto de datos etiquetado para entrenar y evaluar el modelo. Estos datos se consiguieron tanto de forma manual como con ayuda de plataformas como Kaggle. Los datos se procesaron para adecuarlo al uso en un modelo aplicando técnicas de redimensionado y normalización.
- **Entrenamiento:** El modelo se entrenó usando el conjunto de datos, probando distintas configuraciones tratando de obtener la arquitectura óptima
- **Evaluación:** Se evaluaron los distintos modelos realizando mejoras.
- **Análisis de resultados:** Se interpretaron los resultados con distintas formas, dando una visión más clara del comportamiento de la CNN.
- **Puesta en marcha de Raspberry:** Se realizaron todos los procesos necesarios para obtener una Raspberry Pi funcional a partir de una tarjeta microSD vacía. En ella se implementó el sistema desarrollado junto con una cámara y un display para uso en un entorno práctico.

- **Validación del sistema:** Se realizaron pruebas con cada una de las letras del lenguaje de signos inglés, obteniendo información acerca de la precisión y los fallos en la clasificación.
- **Análisis de resultados:** Se representaron los resultados con distintas formas de interpretación para poder hacer el análisis del funcionamiento en el sistema final. Esto permite comprobar las limitaciones y posibles mejoras.

5.2. Reflexiones sobre el trabajo realizado

En este trabajo se han aprendido una gran cantidad de utilidades y dando una fuerte introducción al uso de inteligencia artificial, sus aplicaciones y sus infinitas posibilidades, resultando ser algo muy enriquecedor no solo en términos académicos sino a nivel personal.

Además, se ha aprendido una base de cómo usar distintos softwares dando especial énfasis al uso de Python, GitHub y a trabajar con la consola de comandos, siendo conocimientos muy útiles para el futuro.

Se han obtenido resultados satisfactorios a pesar de los fallos y posibles mejoras que pueden existir.

5.3. Limitaciones y posibles mejoras

Como limitaciones, las mayores barreras que se han encontrado son:

- Las limitaciones en el hardware pues la Raspberry Pi con la que se hizo, como se ha mencionado antes, tenía una RAM muy limitada.
- Partir de un casi absoluto desconocimiento de la mayoría de los softwares y hardware que se han utilizado.

- El tiempo del que se dispuso para realizar el proyecto, evitando que se puedan realizar más pruebas con distintas arquitecturas, u otras herramientas que mejoren la CNN.
- También hay algunas pequeñas limitaciones como por ejemplo que no todos los programas dan soporte para Raspberry, por lo que hay que hacer preparaciones previas a su uso, pero no son de gran complejidad.

Antes se han mencionado algunas soluciones, que por tanto serían posibles mejoras. Además de eso, hay otras mejoras que serían más estéticas como por ejemplo en lugar de representar sólo una letra en el SSD 1306, se podrían guardar las letras en un vector para hacer una palabra y añadir una categoría como señal de fin, haciendo que con las letras que se van identificando se vaya formando una palabra. Finalmente, al recibir la señal de fin se representaría el vector y no solo una letra, permitiendo así identificar palabras habiéndolas deletreado con el lenguaje de signos.

Otra posible mejora es hacer que en el SSD 1306 de indicaciones al usuario de lo que está haciendo, por ejemplo, cuando esté capturando la imagen que aparezca un mensaje diciendo algo como “capturando la imagen, presione ‘tecla’ para hacer la foto”. Con esto se conseguiría que la aplicación sea mas amigable con el usuario, aunque es un detalle más estético que no afecta a la funcionalidad.

También se puede considerar añadir todo el programa a una Raspberry con más potencia, permitiendo la posibilidad de aplicar las soluciones antes expuestas.

6. PROBLEMAS ENCONTRADOS Y TRABAJO FUTURO

6.1. Problemas encontrados

Durante el proyecto se han encontrado muchos problemas de naturalezas distintas, se van a destacar algunos:

- **Datasets:** Antes de decidir crear un programa para crear el dataset, se probaron distintos conjuntos de datos de entrenamiento descargados de Kaggle. Estos datasets daban distintos problemas, alguno daba imágenes de 28x28 pixeles con lo que las manos eran poco reconocibles, otro dataset tenía 3000 imágenes por cada letra, pero

al entrenar el modelo sobre aprendía con lo que alcanzaba una precisión del 100% pero al clasificar imágenes fuera del dataset las clasificaba mal.

- **Implementación de generador de imágenes:** Se intentó implementar un generador de imágenes que añada modificaciones en las fotos para tener una mayor variedad, pero además de servir para identificar el por qué acercar las imágenes o rotarlas empeora el funcionamiento, dio muchos problemas en el procesado ya que el generador "ImageDataGenerator" necesita y devuelve formatos específicos que dificultaban el tratamiento de las imágenes.

- **Uso de GPU:** En el entrenamiento de los modelos tardaba demasiado tiempo, al comprobar los componentes que usaba el programa se pudo observar que no TensorFlow no tenía acceso a la GPU (ver figura 6.1.1). Para solucionar esto se siguieron los pasos que se indican en el foro de TensorFlow. A pesar de seguir la guía en el entorno por defecto de conda seguía dando el mismo problema, por lo que se optó por instalar "miniconda" que es una versión de conda más básicas y se creo el entorno virtual siguiendo los mismos pasos del foro. En este último entorno se arregló el problema.

```
print(tf.__version__)
tf.config.list_physical_devices()
```

```
2.10.0
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```

Figura 6.1.1 TensorFlow no reconoce la GPU

- **Instalación de sistema operativo incorrecto:** Debido a la falta de información, la primera vez que se instaló el sistema operativo Raspbian en la Raspberry que viene como opción por defecto. Este es el que tiene una arquitectura de x32 bits, por lo que al intentar instala TensorFlow generaba constantemente incompatibilidades. La solución fue instalar el sistema operativo correcto, para describir el origen de este error se siguió un tutorial.

- **Alimentación insuficiente de Raspberry:** Inicialmente se tenía una fuente de alimentación que no daba suficiente potencia. Se tuvo que comprar una toma de alimentación que sea capaz de suministrar al menos 3V y 5A.
- **Problema en el conector HDMI:** Inicialmente se intentó acceder al Raspberry usando un monitor, sin embargo, la pantalla aparecía con colores que la hacían difícil de usar y además se cancelaba la conexión. Para solucionarlo se tuvo que acceder a la Raspberry por SSH, de forma remota y usando RealVNC Viewer.
- **Problema al visualizar Raspbian por SSH:** En el primer intento para controlar el microordenador en remoto no se mostraba nada en la pantalla, para solucionarlo se añadieron unas líneas en el archivo congif.txt.

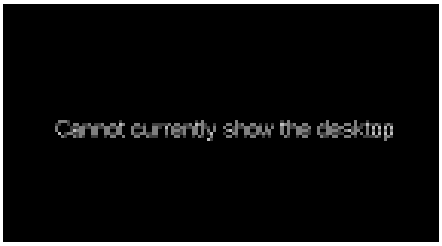


Figura 6.1.2 No se ve la pantalla

```
# uncomment if you get no picture on HDMI for a default "safe" mode
#hdmi_safe=1
hdmi_force_hotplug=1
hdmi_group=2
hdmi_mode=9
```

Figura 6.1.3 Modificación en archivo config.txt

6.2. Trabajo futuro

En este proyecto se ha trabajado con un clasificador de imágenes estáticas y que use una señal para ejecutar la clasificación, como objetivo futuro se va a intentar hacer un programa capaz de trabajar en tiempo real y que en lugar de esperar a la señal de teclado esté constantemente clasificando, de forma que en el SSD 1306 en lugar de escribir una letra estática se represente todo lo que se clasifique.

Además, se puede intentar ampliar conocimiento para trabajar con imágenes en movimiento puesto que, siguiendo en la línea del lenguaje de signos, hay palabras que se transmiten con movimiento por lo que para su clasificación se necesita por aprender a partir de movimientos.

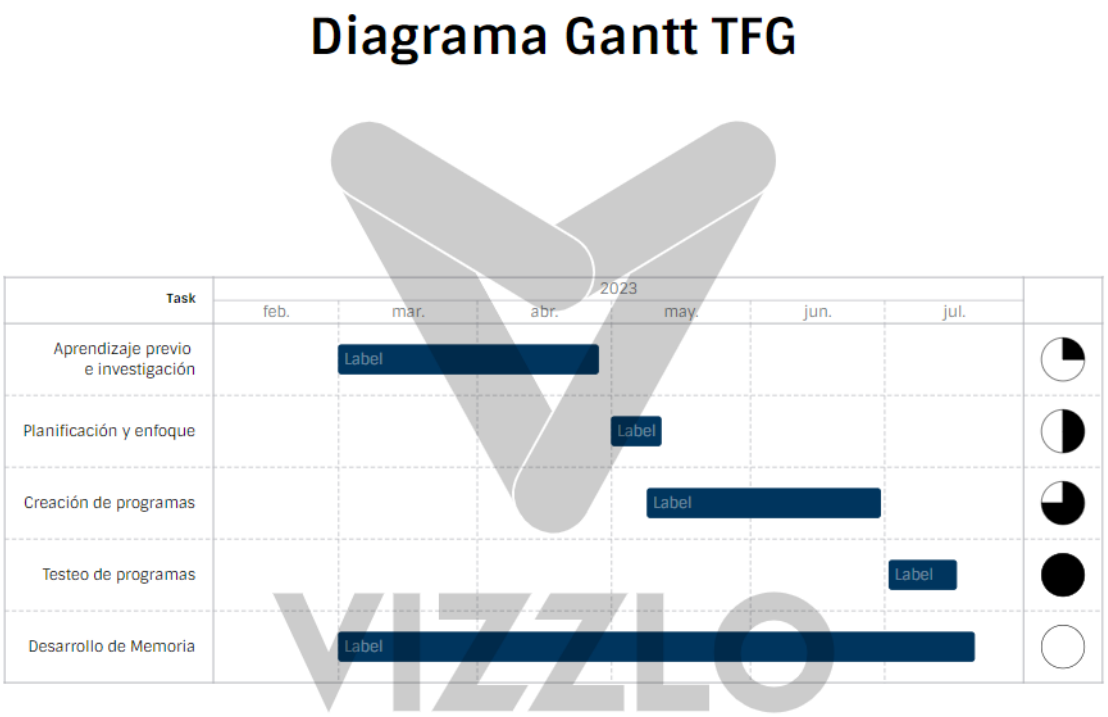
Aplicando otro enfoque, también se puede ampliar conocimiento para clasificar señales que no necesariamente sean imágenes, por ejemplo, se podría clasificar señales de sonido.

También se puede aprender a usar otros tipos de redes, ya que en este proyecto sólo se han usado redes convolucionales, pero existen otras como los árboles de decisión.

7. Diagrama de Gantt y presupuesto

7.1. Diagrama de Gantt

La planificación inicial para el proyecto fue la siguiente:



Sin embargo, en la práctica, el periodo de aprendizaje previo y creación del programa requirió más tiempo, teniendo que aprender a la vez que se iba desarrollando el programa. Además,

ha habido aspectos nuevos de los que se tuvo que aprender que surgieron mientras se desarrollaba el proyecto.

7.2. Presupuesto

Para este proyecto, se necesitan distintos materiales que traen un coste. Se va a detallar cada uno de ellos:

Costes básicos del proyecto

Componente	Precio	Comentarios
Raspberry Pi 4	60-100€	El precio varía en función de la memoria RAM
Alimentación	36€	Coste real en una tienda de electrónica
Cable Ethernet	3€	Precio medio de un cable básico en internet
Tarjeta microSD	8€	Precio en Amazon
Protoboard y cables	15€	Precio en Amazon
Display OLED SSD1306	7€	Precio en Amazon
Cámara	70€	

Esos son los costes para el caso de este proyecto, en el cual se hizo la conexión remota por SSH, si no se hubiese hecho en remoto habría que añadir:

Componente	Precio	Comentarios
Teclado	10€	Coste real en una tienda de electrónica
Mouse	12€	Coste real en una tienda de electrónica
Cable HDMI-microHDMI	9€	Precio medio de un cable básico en internet
Pantalla o monitor	-	En este caso el precio es muy variado según lo que desee el usuario.

Hay que añadir que, por suerte, para la realización de este proyecto ya se disponía de la mayoría de los elementos. Lo que se tuvo que adquirir nuevo fueron la protoboard y cables, el display OLED SSD1306 y la alimentación de la Raspberry.

En cuanto a la mano de obra, considerando que un ingeniero cobre aproximadamente 25€ la hora, y teniendo en general aproximadamente 352 horas (8h/día durante 2 meses) juntando todo lo que se ha dedicado se tendría un coste de mano de obra de 8800€

8. Referencias bibliográficas

Adafruit Industries. (s. f.). GitHub. Recuperado 20 de julio de 2023, de https://GitHub.com/adafruit

Amezcua Aguilar, T., & Amezcua Aguilar, P. (2018). Contextos inclusivos: El reconocimiento de la lengua de signos como derecho de las personas con diversidad funcional. *Index.comunicación: Revista científica en el ámbito de la Comunicación Aplicada*, *8*(1), 123-148.

Anwarul, S., & Joshi, D. (2020). *Deep Learning With TensorFlow* (pp. 96-120). https://doi.org/10.4018/978-1-7998-3095-5.ch004

Arias, V., & Manuel, J. (s. f.). *Modelo de aprendizaje profundo/red neuronal convolucional (CNN) para clasificación de calidad de ácidos grasos por imágenes de semillas de Helianthus annuus*.

Ayora, M. J. M. (s. f.). *Uso de redes neuronales para identificación de matrículas*.

Bishop, C. M. (s. f.). *Neural Networks for Pattern Recognition*.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Chollet, F. (2018). *Deep learning with Python*. Manning Publications Co.

clueple c (Director). (2019, junio 3). *Install Jupyter Lab | Linux Or Raspbian| Set Dark Theme | 2019*. https://www.youtube.com/watch?v=gV0DtmWYpLA

codebasics (Director). (2020, noviembre 1). *Data augmentation to address overfitting | Deep Learning Tutorial 26 (TensorFlow, Keras & Python)*. https://www.youtube.com/watch?v=mTVf7BN7S8w

DM Sans. (s. f.). Google Fonts. Recuperado 20 de julio de 2023, de https://fonts.google.com/specimen/DM+Sans

EvidenceN (Director). (2021, enero 20). *How to Build and Interpret Confusion Matrix Using Python & Sklearn*. https://www.youtube.com/watch?v=8gLewErTU24

Giovanni, R. I. I. (s. f.). *Implementación de reconocimiento facial y visión artificial en robot nao con Python y OpenCV*.

Google/mediapipe. (2023). [C++]. Google. https://GitHub.com/google/mediapipe (Obra original publicada en 2019)

Hand landmarks detection guide | MediaPipe. (s. f.). Google for Developers. Recuperado 20 de julio de 2023, de https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

Image Module. (s. f.). Pillow (PIL Fork). Recuperado 20 de julio de 2023, de https://pillow.readthedocs.io/en/stable/reference/reference/Image.html

Instalar TensorFlow con pip. (s. f.). TensorFlow. Recuperado 20 de julio de 2023, de https://www.TensorFlow.org/install/pip?hl=es-419

Interfacing-circuit-diagram-of-OLED-Display-with-Raspberry-Pi.png (700×932). (s. f.). Recuperado 20 de julio de 2023, de https://circuitdigest.com/sites/default/files/circuitdiagram_mic/Interfacing-circuit-diagram-of-OLED-Display-with-Raspberry-Pi.png

Jiménez, Y. A. (s. f.). *Clasificación de Flores con Redes Neuronales Convolucionales*.

Keating, E., & Mirus, G. (2003). American Sign Language in virtual space: Interactions between deaf users of computer-mediated video communication and the impact of technology on language practices. *Language in Society*, *32*(5), 693-714. https://doi.org/10.1017/S0047404503325047

Krish Naik (Director). (2019, diciembre 21). *Tutorial 26- Create Image Dataset using Data Augmentation using Keras-Deep Learning-Data Science*. https://www.youtube.com/watch?v=hxLU32zhze0

Larranaga, P., Inza, I., & Moujahid, A. (2023). *Tema 8. Redes Neuronales*.

López-Saca, F. (2019). *Clasificación de imágenes usando redes neuronales convolucionales*. http://zaloamati.azc.uam.mx//handle/11191/6123

Ltd, R. P. (s. f.). *Raspberry pi 4*. Raspberry Pi. Recuperado 20 de julio de 2023, de https://www.Raspberrypi.com/products/Raspberry-pi-4-model-b/

Martín, J. S.-B., Delgado, A. G., & de, J. (s. f.). *Aforador de vehículos en carretera mediante OpenCV sobre plataforma local.*

Nicholas Renotte (Director). (2022, abril 25). *Build a Deep CNN Image Classifier with ANY Images*. https://www.youtube.com/watch?v=jztwpsIzEGc

Nonaka, A. M. (2004). The forgotten endangered languages: Lessons on the importance of remembering from Thailand's Ban Khor Sign Language. *Language in Society*, *33*(5), 737-767. https://doi.org/10.1017/S004740450404504X

OpenCV. (s. f.). OpenCV. Recuperado 20 de julio de 2023, de https://OpenCV.org/

Ortiz, D. C., & Jiménez, F. J. F. (s. f.). *Inteligencia Artificial con TensorFlow para predicción de comportamientos*.

Quer, J., & Steinbach, M. (2019). Handling Sign Language Data: The Impact of Modality. *Frontiers in Psychology*, *10*. https://www.frontiersin.org/articles/10.3389/fpsyg.2019.00483

Ringa Tech (Director). (2021a, julio 17). *Tu primer clasificador de imágenes con Python y TensorFlow*. https://www.youtube.com/watch?v=j6eGHROLKP8

Ringa Tech (Director). (2021b, agosto 16). *Redes Neuronales Convolucionales—Clasificación avanzada de imágenes con IA / ML (CNN)*. https://www.youtube.com/watch?v=4sWhhQwHqug

Ringa Tech (Director). (2021c, septiembre 14). *Crea un clasificador de perros y gatos con IA, Python y TensorFlow—Proyecto completo*. https://www.youtube.com/watch?v=DbwKbsCWPSg

Sam Westby Tech (Director). (2022, marzo 15). *How to Install TensorFlow 2 and OpenCV on a Raspberry Pi*. https://www.youtube.com/watch?v=vekblEk6UPc

Sistemas Inteligentes (Director). (2020, julio 14). *Data Augmentation para clasificación de imágenes usando keras—TensorFlow (Parte 1)*. https://www.youtube.com/watch?v=4G2WAgJ4G3Y

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

TensorFlow Forum. (s. f.). TensorFlow Forum. Recuperado 20 de julio de 2023, de https://discuss.TensorFlow.org/

TensorFlow-bin/previous_versions at main · PINTO0309/TensorFlow-bin. (s. f.). GitHub. Recuperado 20 de julio de 2023, de https://GitHub.com/PINTO0309/TensorFlow-bin/tree/main/previous_versions

Tf.keras.preprocessing.image.ImageDataGenerator | TensorFlow v2.13.0. (s. f.). TensorFlow. Recuperado 20 de julio de 2023, de https://www.TensorFlow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Thakur, A. (2019, julio 20). *American Sign Language Dataset*. https://www.kaggle.com/datasets/ayuraj/asl-dataset

Universitat Politècnica De València, E. (2014). Universitat Politècnica de València. *Ingeniería del agua*, *18*(1), ix. https://doi.org/10.4995/ia.2014.3293

Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.-L., & Grundmann, M. (2020). *MediaPipe Hands: On-device Real-time Hand Tracking* (arXiv:2006.10214). arXiv. http://arxiv.org/abs/2006.10214

BETANCOURT, G. . A. . (2005). LAS MÁQUINAS DE SOPORTE VECTORIAL (SVMs). Scientia Et Technica, 1(27). https://doi.org/10.22517/23447214.6895