ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento de Tecnología Electrónica

Tecnología Electrónica

TRABAJO FIN DE GRADO

Sistema robotizado basado en visión artificial para envasado de hortalizas

Grado en: Ingeniería Electrónica Industrial

Autor: Gálvez Ortiz, Noelia

Tutor: Francisco José Sánchez Pacheco

Cotutor:

MÁLAGA, junio de 2.023

Agradecimientos:

Gracias a mi tutor Francisco, por haberme dado la oportunidad de realizar este trabajo y haber confiado en mí. Eres un ejemplo a seguir.

A Grupo Premo y a todos sus integrantes, en especial a Maher, por su amabilidad y paciencia.

Después de estos años, se me hace complicado resumir en pocos párrafos como agradecer a quienes han estado conmigo en este camino.

Mi familia, lo primero siempre. Gracias, gracias y mil gracias por darme la oportunidad de hacerlo posible y apoyarme siempre. Silverio, eres la persona que más me ha enseñado en esta vida, y eso que eres mi hermano pequeño, bueno, ya no tan pequeño.

A mi abuela, por tantísimas velas que me ha encendido deseándome, de corazón, toda la suerte del mundo.

A mi tita Encarni.

A mis amigas y amigos de Málaga y Varsovia. Puedo escribir durante horas sobre vosotros, pero me conformo con decir que tener vuestro apoyo incondicional es un tesoro que nunca quiero perder. Por creer en mí y empujarme hacia mis metas, habéis hecho que el camino sea fácil hasta cuando parecía hundirse, quedo en deuda con vosotros por haberme ayudado tanto.

A Golon y Mv, me habéis enseñado el verdadero significado de la palabra amistad.

A Víctor, gracias por ser y estar.

A Miguelito, tendría que hacer un TFG solo para explicar lo mucho que te quiero.

Soy una afortunada de estar tan bien rodeada, y espero, por difícil que parezca, estar a vuestra altura para hacéros lo feliz que me hacéis a mí.

Resumen

En el siguiente trabajo fin de grado se desarrolla un sistema robotizado basado en visión artificial para envasado de hortalizas, donde, mediante imágenes obtenidas por webcam, se clasificará las hortalizas en función de su tamaño. Se diseñará un gripper que garantice el agarre de estas sin dañarlas, y se colocarán en distintos lugares mediante un brazo robot dependiendo de la clasificación obtenida anteriormente.

Se decide realizar este proyecto con el fin de agilizar el envasado de hortalizas y verduras en plantas que aún siguen realizando este proyecto manualmente.

Palabras clave

Visión artificial, inteligencia artificial, CNN, robot colaborativo, gripper, calabacines.

Abstract

The following thesis Project develops a vision-based robotics system for vegetable packaging, where vegetables will be classified based on their size using images captured by a webcam. A gripper is designed to ensure a secure hold of the vegetables without causing any damage, and they will be placed in different locations using a robotic arm depending on the previous classification obtained.

This project is undertaken with the aim of streamlining the packaging process of vegetables in industrial plants that still rely on manual labor.

Keywords

Computer vision, artificial intelligence, CNN, collaborative robot, gripper, zucchinis.

Índice

Índice de figuras

Índice de códigos

Índice de tablas

1. Introducción

La inteligencia artificial se encarga de proporcionar el marco teórico y las herramientas necesarias para desarrollar sistemas inteligentes que puedan procesar, analizar y comprender imágenes y videos. Los algoritmos y modelos de aprendizaje automático, en particular el aprendizaje profundo, han demostrado ser especialmente efectivos en el campo de la visión artificial.

La visión artificial utiliza algoritmos y técnicas de inteligencia artificial para permitir a las máquinas interpretar y comprender el contenido visual, de manera similar a como lo hacen los seres humanos. [1]

Mediante el uso de redes neuronales convolucionales (a la cual se hará referencia a partir de ahora como CNN, del inglés *Convolutional Neural Network*), la visión artificial puede realizar tareas como el reconocimiento de objetos, la detección y seguimiento de personas, la segmentación de imágenes y la generación de descripciones automáticas.

En los últimos años, la industria agroalimentaria ha experimentado una transformación sin precedentes gracias a esta incorporación de tecnologías innovadoras, que están permitiendo la automatización y optimización de los procesos de producción y envasado de alimentos. En este contexto, la automatización de las líneas de procesado de hortalizas se presenta como una oportunidad para mejorar la eficiencia y reducir los costes, al mismo tiempo que se garantiza una mayor calidad y seguridad alimentaria haciendo uso junto de los robots y robots colaborativos.

Los robots con visión artificial han revolucionado diversos campos y sectores, permitiendo la automatización de tareas que antes solo podían ser realizadas por seres humanos. Estos robots, equipados con sistemas de visión artificial basados en algoritmos de inteligencia artificial, son capaces de percibir y comprender su entorno visual de manera similar a como lo hacen los seres humanos. A los robots se les proporciona así la capacidad de capturar imágenes o videos de su entorno utilizando cámaras, y luego procesar y analizar esa información visual para tomar decisiones y realizar acciones específicas [2]. Muchas de sus aplicaciones se encuentran en diversos sectores, como la industria manufacturera, logística, medicina… Sin embargo, uno de los campos que más se está desarrollando es en el sector de la alimentación, haciendo que disfrute de numerosas ventajas como [3]:

- Optimización de procesos: Con el uso de máquinas que no reducen su eficiencia aun realizando tareas repetitivas durante horas.
- Mejora de procesos de calidad: la idoneidad de un producto puede ser calificado mediando el uso de visión artificial.
- Automatización de etapas del proceso de producción: Un robot con inteligencia artificial puede realizar actividades que son peligrosas evitando así la mano de obra humana.

2. Objetivos del trabajo

El objetivo principal del presente trabajo es desarrollar e implementar un sistema robotizado basado en visión artificial para envasado de hortalizas, centrándose específicamente en la detección del tamaño de calabacines. Para lograr esto, se emplea una Red Neuronal Convolucional (CNN) entrenada en una base de datos creada exclusivamente para este proyecto.

Esta base de datos recogerá las imágenes de los tres calibres por longitud contemplados según el Reglamento (CEE) nº 1292/81 que serán [4]:

- Pequeño, de 7 cm a 14 inclusive.
- Mediano, de 14 cm exclusive a 21 cm inclusive.
- Grande, de 21 cm exclusive a 30cm.

Después de la clasificación, los calabacines serán colocados en tres destinos distintos utilizando un brazo robótico colaborativo (Cobot) de seis grados de libertad. Para facilitar este transporte, se ha diseñado una pinza (a partir de ahora, gripper) especializada capaz de garantizar una adecuada sujeción de los calabacines.

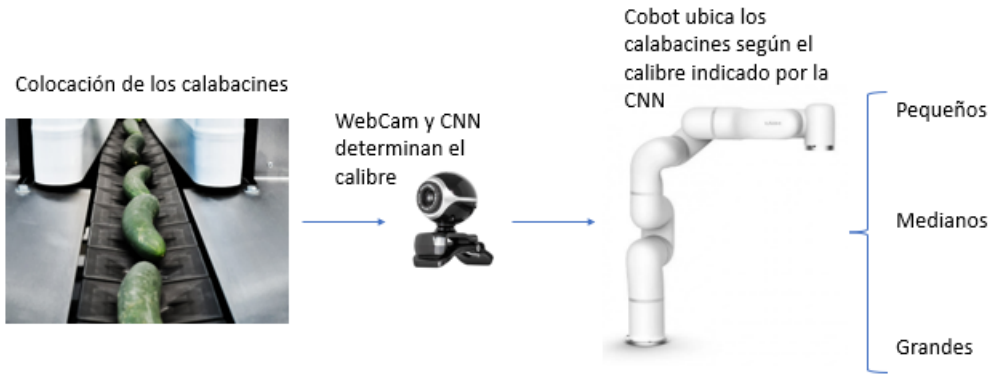


Figura 1: Esquema del proyecto

La implementación de este sistema permitirá automatizar y optimizar el proceso de envasado de calabacines, reduciendo la necesidad de una intervención humana directa en el proceso de clasificación. Este proyecto está basado en la experiencia de la autora en Hortoventas, una empresa ubicada en Ventas de Zafarraya que se dedicada al envasado manual de diversas hortalizas, entre ellas calabacines.

3. Marco teórico de la Inteligencia Artificial

En esta sección se profundiza en las disciplinas de visión artificial, inteligencia artificial, aprendizaje máquina y aprendizaje profundo, siendo esta última la herramienta más adecuada para lograr una eficiente clasificación de imágenes.

Como se ha descrito anteriormente, la visión artificial es una rama especifica de la IA que se centra en el procesamiento y análisis de información visual por parte de las máquinas. Su objetivo es permitir a los sistemas automatizados comprender y extraer significado de las imágenes y videos de manera similar a como lo hacen los seres humanos.

El aprendizaje máquina (a partir de ahora ML, del inglés *Machine Learning*) es una disciplina de la IA, que se enfoca en el aprendizaje automático. Al adentrarse en el ML, se encuentra el aprendizaje profundo (en adelante, DL del inglés *Deep Learning*), un conjunto de algoritmos que permiten un mayor nivel de abstracción y requieren una menor supervisión humana para llevar a cabo estas tareas inteligentes.

Sin embargo, lo que hace posible obtener la profundidad son las redes neuronales, sistemas computacionales formados por múltiples capas interconectadas que trabajan juntas para alcanzar el nivel de profundidad deseado. Estas redes neuronales contienen neuronas artificiales que imitan el funcionamiento de las células nerviosas biológicas y permiten el flujo de información a través de ellas. La visión artificial utiliza algoritmos y técnicas como las CNN para realizar tareas de reconocimiento de objetos, la detección de rostros, segmentación de imágenes y muchas otras aplicaciones visuales.

La combinación de la visión artificial con el ML y el DL ha impulsado avances significativos en el sector agrario. Estas técnicas permiten a las máquinas adquirir una compresión más profunda de las imágenes y realizar tareas sofisticadas.

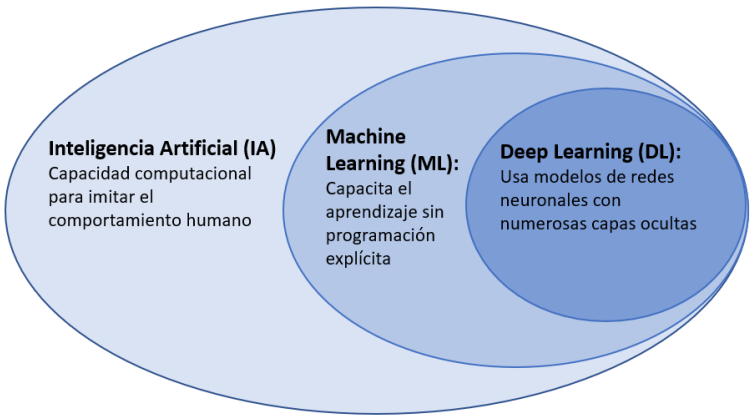


Figura 2: Evolución de la IA, ML y DL

3.1. Contexto histórico

En este subcapítulo, trata de situar temporalmente la inteligencia artificial y la visión artificial.

La historia de la IA se remonta a la antigüedad, donde los seres humanos ya intentaban crear autómatas que se movieran y realizaran tareas simples. Sin embargo, la IA como campo de estudio comenzó en la década de 1950, cuando se celebró una conferencia en la Universidad de Dartmouth, Estados Unidos, donde se acuño el término "inteligencia artificial" y se reunieron algunos de los primeros investigadores en el campo [5]

En los primeros años de aparición de la IA, se centraron en la creación de programas que pudieran resolver problemas simples, como el juego de damas o el ajedrez. Uno de los primeros programas de IA exitosos fue el "The Logic Theorist" de Allen Newell y Herbert A.Simon, que podía demostrar teoremas matemáticos.

En la década de 1960, se desarrollaron las redes neuronales artificiales, que imitaban la estructura del cerebro humano y se utilizan en la actualidad para el Deep Learning. Al mismo tiempo, surgió el concepto de visión artificial, con el objetivo de conectar una cámara de vídeo a un computador

En 1963, los computadores fueron capaces de convertir imágenes bidimensionales en formas tridimensionales. Larry Roberts, creador de ARPAnet, realizó un trabajo pionero en visión artificial en 1961 al desarrollar un programa llamado "Mundo de Microbloques", donde un robot podía "ver" una estructura de bloques sobre una mesa y analizar su contenido desde otra perspectiva. [6]

Además, en la década de 1960, la IA comenzó a usarse para resolver el problema de visión artificial. Con el tiempo, se fueron creando aplicaciones de reconocimiento fácil y se avanzó en el reconocimiento de objetos.

En la década de 1970, la IA experimentó una desaceleración debido a la falta de progreso e inversiones en el desarrollo. Sin embargo, en1980 se crearon nuevas técnicas de aprendizaje automático, como los algoritmos genéricos y los árboles de decisión, que permitieron a las computadoras tomar decisiones por sí mismas. Esta etapa, se centra en un ML muy superficial, es decir, en redes neuronales muy simples que después ajustan los parámetros de forma automática a partir de los datos de entrenamiento. [6]

En 1997, la historia de la IA estuvo marcada por un suceso importante. La inteligencia artificial ganó al campeón mundial de ajedrez, Gary Kasparov. Por primera vez, el hombre ha sido vencido por una máquina. [7]

Figura 3: El campeón mundial enfrentándose a la inteligencia artificial [7]

En 2008, Google hizo accesible la inteligencia artificial al público través del reconocimiento de voz en los smartphones.

En resumen, la historia de la IA es una historia de desafíos, progresos y retrocesos, pero a medida que la tecnología ha avanzado, la IA ha demostrado su potencial para transformar el mundo en el que vivimos.

3.2. Introducción al Machine Learning (ML)

En el contexto de la visión artificial, el Machine Learning es fundamental para proporcionar métodos para el reconocimiento, clasificación y compresión de objetos y patrones visuales en imágenes.

El Machine Learning (ML) es una subdisciplina de la IA que se enfoca en dotar a las máquinas de la capacidad de aprender de forma automática a partir de aprender de forma autónoma a partir de emulaciones del entorno adyacente. A diferencia de programar una máquina para realizar una tarea especifica, el ML busca que la máquina aprenda a realizar una tarea por sí misma [8] [9].

Una de las aplicaciones más características del ML es la clasificación de imágenes, la cual puede requerir distintos grados de supervisión humana dependiendo del método empleado. Estos métodos se fraccionan en tres subgrupos: el aprendizaje supervisado, el aprendizaje débilmente supervisado y el aprendizaje no supervisado. Se detallan a continuación:

- El aprendizaje supervisado consiste en entrenar a un algoritmo a relacionar las variables entradas con las de salida haciendo uso de ejemplos previamente etiquetados. En este proceso, un ser humano supervisa el algoritmo y ajusta sus parámetros para mejorar su rendimiento.
- El aprendizaje no supervisado busca descubrir patrones en los datos de entrada sin necesidad de etiquetado previo ni supervisión humana.
- El aprendizaje débilmente superado utiliza información adicional, como etiquetas de textos o metadatos, para obtener información sobre las categorías, sin necesidad de conocimiento fundamental. Aunque se encuentra en estudio, este tipo de aprendizaje puede resultar prometedor en el futuro.

En el ejemplo de la Figura 4: Supervisión supervisado y no supervisado, se muestra una clasificación a través del método supervisado y no supervisado. Con la supervisión, el resultado del problema es enseñar a identificar los tres posibles calibres, donde G se refiere a grande, M a mediano y P a pequeño, la red aprende a hacer la clasificación que el usuario le ha indicado previamente con etiquetas, por esa razón conoce el nombre de los elementos [10] [11].

Por otro lado, si se decide usar el aprendizaje no supervisado para clasificar imágenes, se cuenta únicamente con las imágenes de entrada sin ningún tipo de información adicional. En este caso, el algoritmo buscará patrones de similitud en las formas y colores de las imágenes para agruparlas en categorías. Aunque el algoritmo detecte patrones distintos entre las imágenes, en este caso no se conoce el nombre de los elementos que están siendo clasificados.

En este proyecto se ha utilizado un aprendizaje supervisado.

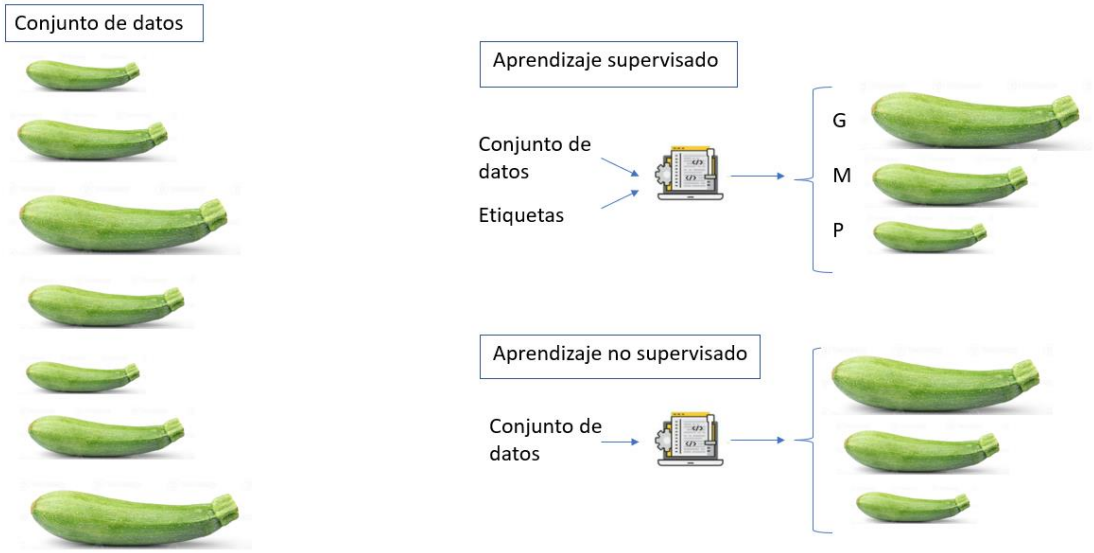


Figura 4: Supervisión supervisado y no supervisado.

Las redes neuronales, y, por tanto, el Deep Learning están basados en el lenguaje supervisado, aunque un grado más débil que otros algoritmos de ML.

En el siguiente apartado, se introduce el DL.

3.3. Introducción a Deep Learning (DL).

Para comprender que es el Deep Learning es importante distinguirlo de otras disciplinas dentro del campo de la IA. En el ML, el usuario es responsable de extraer las características de los datos de entrada, lo que requiere la ayuda de un operador humano para proporcionar miles de ejemplos de entrenamiento y corregir los errores. En cambio, el DL es un subcampo del aprendizaje automático que ya incluye la extracción de características como parte del proceso [12]

El DL ha surgido como una herramienta poderosa para abordar tareas complejas de visión artificial.

Para clasificar imágenes utilizando técnicas de DL, se le proporciona a la red una imagen con una etiqueta que indica a que clase pertenece. La red está compuesta por múltiples capas de nodos interconectados y, cada entrada de la red es un píxel. El algoritmo es el encargado de encontrar los patrones y peculiaridades de las categorías sin la participación humana directa. A diferencia del ML, que se basa en la lógica lineal, el DL se basa en teorías sobre el funcionamiento del cerebro humano. Después de cada ciclo, la red aprende ajustando las conexiones o pesos entre las neuronas para minimizar el error [13].

4. Estado del arte de la visión artificial

La visión artificial ha experimentado numerosos avances que la han convertido en una herramienta ampliamente utilizada en diversos campos en la actualidad. Estos avances permiten la captura de información visual, su análisis en tiempo real, así como el almacenamiento de datos para su posterior análisis o revisión.

Entre los campos más utilizados se destacan los siguientes:

- Medicina: en el ámbito de la medicina, la visión artificial ha revolucionado la clasificación y visualización de imágenes médicas, permitiendo una mejor comprensión y diagnóstico. Con el procesamiento de imágenes de alta resolución, se ha facilitado la realización de cirugías remotas, eliminando la necesidad de que un/a médico/a especializado esté físicamente presente en el lugar de la operación [14]
- Automoción: La visión artificial desempeña un papel crucial en la conducción autónoma, junto con otros tipos de sensores, y su uso se extiende tanto en la actualidad como en el futuro. Es fundamental para la detección de carriles, objetos y seres humanos, lo que permite que el vehículo tome decisiones basadas en su entorno.

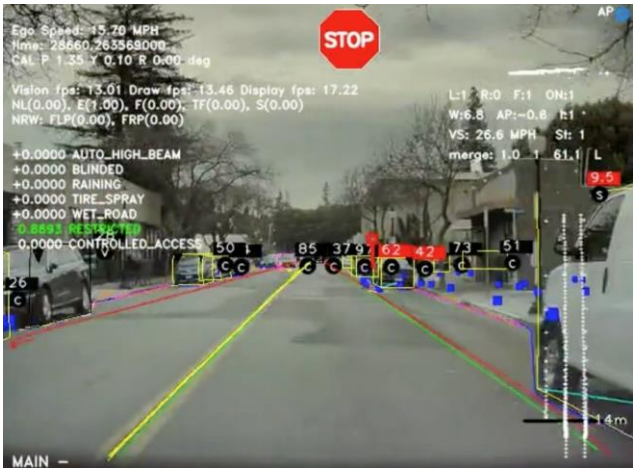


Figura 5: Autopilot de Tesla [15]

- Detección de defectos por visión artificial en conexiones electrónicas: la visión artificial se utiliza para comprobar las conexiones electrónicas. Mediante el uso de una cámara y la captura de imágenes, se verifica que las conexiones de diversos aparatos electrónicos estén realizadas correctamente. Se realiza una comprobación para asegurarse de que no falte ningún componente, y comprueba además que están en la correcta ubicación. En el caso de detectar un defecto, se notifica de forma rápida utilizando diversos métodos y, en algunos casos, se puede automatizar el proceso [16].
- Proyecto OPTINVAS utiliza la visión artificial para diferenciar entre el magro y la grasa del producto.

El campo de la visión artificial ha experimentado avances significativos en los últimos años, impulsados por los avances en algoritmos de aprendizaje profundo, la disponibilidad de grandes conjuntos de datos etiquetados y el aumento de potencia computacional.

En el ámbito de reconocimiento de objetos, los modelos de aprendizaje profundo, como las redes neuronales convolucionales (CNN), han alcanzado niveles de precisión sorprendentes en

tareas como la clasificación de imágenes y la detección de objetos. Los enfoques basados en detección de objetos, como Faster R-CNN [18] y YOLO (You Only Look Once) [19], han mejorado la detección en tiempo real y la segmentación de objetos en imágenes y videos.

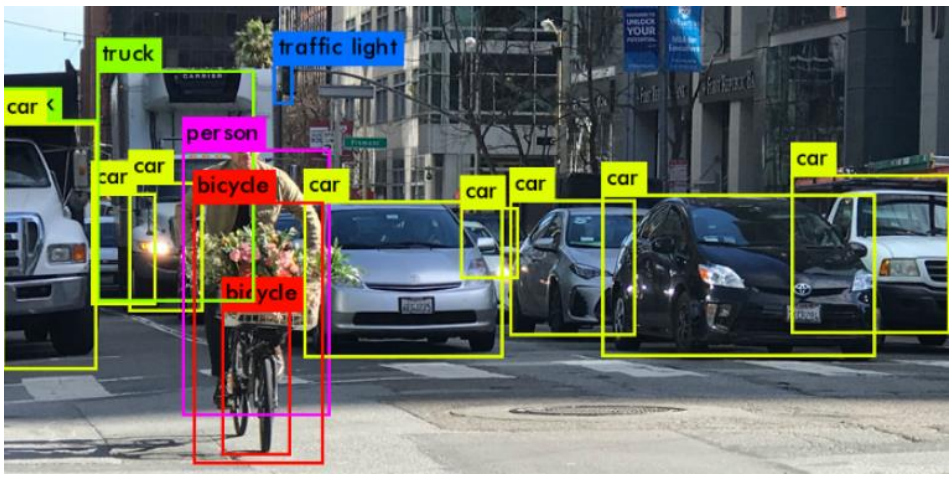


Figura 6: Detección de imágenes con YOLO [20]

5. Las Redes Neuronales Artificiales

Las redes neuronales son unos de los conjuntos de algoritmos de Machine Learning más populares en la actualidad, gracias a los avances tecnológicos que permiten llevar a cabo la gran cantidad de operaciones aritméticas necesarias. El uso de GPUs, o unidades de procesamiento gráfico, ha optimizado el procesamiento en paralelo, lo que se traduce en mayor facilidad, rapidez y un menor coste [21]

Las primeras redes neuronales artificiales, también denominadas ANNs, creadas no podían detectar patrones complejos, ya que sólo interactuaban con un número limitado de neuronas simultáneamente. Por esta razón, pasaron a un segundo plano en la IA. Sin embargo, en la década de los 80 resurgió el interés en esta disciplina al descubrir los modelos de Deep Learning, que permiten aumentar el número de capas ocultas intermedias y, por lo tanto, detectar patrones más complejos. En la actualidad, las redes neuronales desempeñan un papel fundamental en el campo de la IA [22].

5.1. Conceptos básicos

Las redes neuronales artificiales son capaces de aprender de manera adaptable y jerárquica . En las capas iniciales, se aprende a procesar los conceptos más básicos, mientras que en las capas posteriores se construye información más compleja a partir de las salidas de la capa previa que se consideran como entradas. No hay un límite máximo establecido de capas que se pueden agregar a una ANN, y es precisamente este concepto el que da lugar a la profundidad de la red [23].

Se muestra en la siguiente figura, una red neuronal multicapa, donde la capa de entrada y la de salida siempre están presentes. Entre ambas, se insertan capas escondidas, en este ejemplo dos, con conexiones unidireccionales hacia delante.

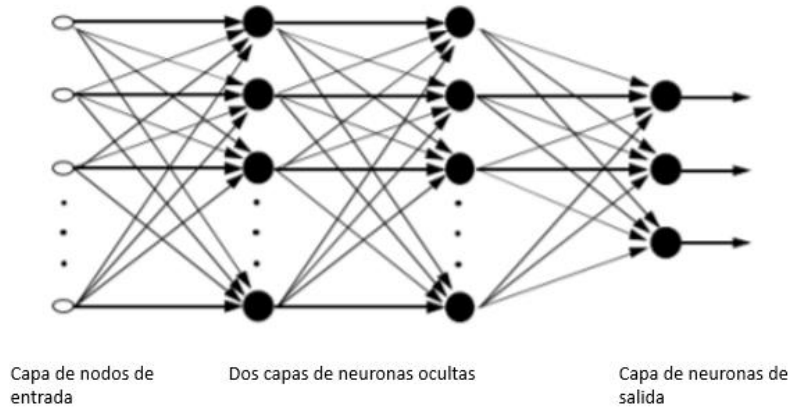


Figura 7: Red neuronal artificial multicapa [24]

5.1.1. Descripción de una neurona biológica

Se puede comenzar con una breve explicación sobre el funcionamiento de las neuronas biológicas, que son elementos más importantes del sistema nervioso humano. Para hacer una asociación con las neuronas artificiales.

Las neuronas biológicas consisten en un cuerpo celular o soma, unido a multiples extensiones llamadas dendritas, que reciben las entradas o impulsos. En el otro extremo del soma se encuentra una exterior final de salida llama axón. Las neuronas están conectadas entre sí a través de la sinapsis, que conecta el axón de una neurona con las dendritas de la siguiente [25] [26]. Puede verse en la siguiente figura

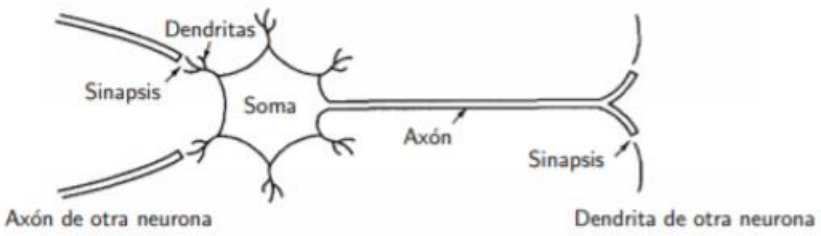


Figura 8: Partes de una neurona biológica [27]

En el tema de las redes neuronales artificiales, estas se componen de capas interconectadas de neuronas. Las conexiones entre las neuronas se asocian con pesos sinápticos, y estas redes también tienen una estructura jerárquica en la que las primeras capas procesan las características más básicas y las capas posteriores procesan información cada vez más compleja.

Con la explicación previa acerca de la estructura y funcionamiento de una neurona biológica, se puede comprender de forma más sencilla cómo funciona una neurona en una red neuronal artificial. En este caso, la neurona es considerada la unidad básica de procesamiento, la cual cuenta con una conexión de entrada que recibe estímulos. Estos estímulos activan la neurona a través de una función de activación y se realiza un cálculo interno. Al finalizar este cálculo, se generan unos valores de salida que pueden ser transmitidos a otra neurona interconectada a la red.

5.1.2. Perceptrón

El perceptrón es la red neuronal más sencilla Solo puede resolver problemas booleanos lineales, por lo que, para abordar modelos más complejos, es necesario aumentar el número de neuronas y apilar más capas intermedias, lo que da lugar al perceptrón multicapa (MLP).

En la siguiente figura, se expone un esquema de la arquitectura de una simple neurona o perceptrón y la red resultante al unir varias de ellas, así como una comparación con su estructura artificial.

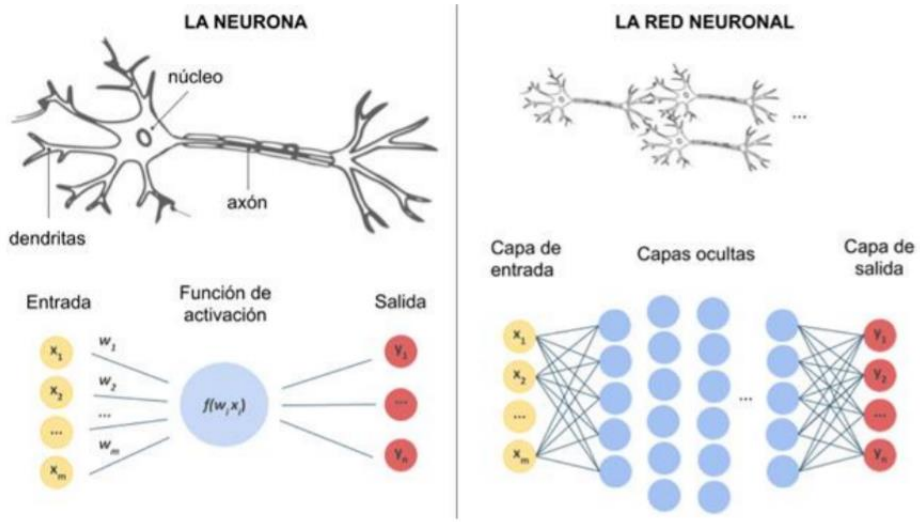


Figura 9: Arquitectura de una neurona y una red neuronal

El perceptrón utiliza la información de entrada para calcular una suma ponderada, basada en los pesos del modelo que determinan el impacto de cada variable de entrada en la neurona. Estos pesos se ajustan reiteradamente para minimizar el error de clasificación entre el resultado esperado y el obtenido. Además, la neurona tiene un sesgo o bias, que es una constante utilizada para evitar multiplicaciones por cero y se controla ajustando su valor. En una neurona con salida binaria, el resultado numérico continuo se evalúa con respecto a un umbral, y el sesgo se utiliza para igualar la expresión a cero. Las variables de entrada pueden ser binarias o un vector de muchas muestras, y la primera capa determina el tamaño inicial de la red. El perceptrón es limitado y solo puede resolver problemas booleanos lineales, por lo que se requiere el uso de perceptrones multicapa para modelos más complejos.

5.1.3. Descenso del gradiente

El siguiente concepto que se detalla a continuación, se denomina descenso de gradiente y se utiliza para ajustar los pesos y sesgos de las neuronas en cada capa de la red. Este algoritmo de ML conocido es un proceso iterativo que busca minimizar una función de coste mediante el cálculo de la derivada parcial de la función, que representa la pendiente o gradiente. Inicialmente, los coeficientes se asignan de manera aleatoria y se recalculan en cada iteración, reduciéndose con el negativo de la derivada y siguiendo el ritmo marcado por una tasa de aprendizaje que indica el tamaño de paso. Luego, se multiplican por la derivada para alcanzar los mínimos locales después de varias iteraciones. El proceso se detiene cuando converge al valor mínimo de la función y no se puede decrementar más, es decir, cuando se alcanza el óptimo, se puede ver el ejemplo en la siguiente figura:

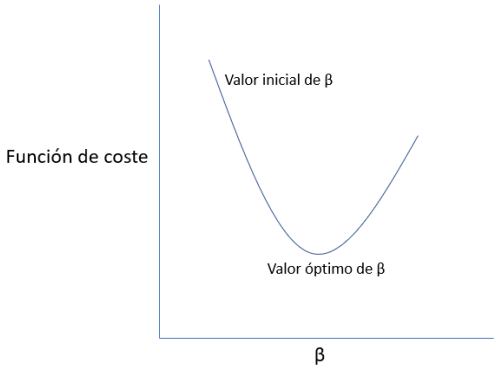


Figura 10: Método del descenso del gradiente

Sin embargo, la técnica de descenso del gradiente tiene una desventaja: si la tasa de aprendizaje es demasiado rápida, puede pasar por alto el verdadero mínimo local para optimizar el tiempo, mientras que, si se elige una tasa demasiado lenta, la función puede quedarse en bucle tratando de encontrar el mínimo local exacto y no llegar a converger.

La tasa de aprendizaje afecta directamente al valor del mínimo que se alcanza y a que velocidad se alcanza. Por lo tanto, se recomienda practicar la alteración de este valor hasta encontrar uno que no ralentice demasiado el proceso y disminuya suficiente el error [28] [29].

Actualmente, existen herramientas para calcular la tasa de aprendizaje de forma dinámica según convenga. Se rige por un principio sencillo: cuando la pendiente de la función coste tiene mucha pendiente, se avanza con una tasa alta, suponiendo que el mínimo no está cerca, para avanzar más rápido. A medida que la curva se suaviza, los pasos se vuelven más pequeños, ya que el mínimo está cerca.

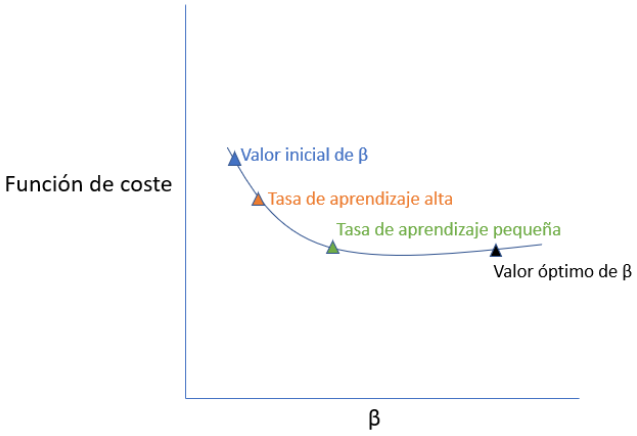


Figura 11: Descenso del gradiente con tasa de aprendizaje

La entropía cruzada categórica está directamente relacionada con el algoritmo de descenso de gradiente utilizado para optimizar los modelos de aprendizaje automático.

Se define así, la entropía cruzada categórica o "categorical_cossentropy" como una función de pérdida comúnmente utilizada en problemas de clasificación cuando se trabaja con variables objetivo categóricas, es decir, el descenso del gradiente se aplica para encontrar los valores óptimos de los pesos y sesgos del modelo que minimizan la entropía cruzada categórica en el conjunto de datos de entrenamiento. A media que se realizan iteraciones del descenso del gradiente, los parámetros del modelo se actualizan gradualmente para mejorar la calidad de las predicciones y reducir la entropía cruzada categórica.

La entropía cruzada categórica mide la diferencia entre la distribución de probabilidad predicha por el modelo y la distribución de probabilidad real de las clases. En esencia, penaliza al modelo cuando la distribución de probabilidad predicha difiere de la distribución real de manera significativa [30].

Para un problema de clasificación de N clases, la fórmula utilizada es:

$$CCE = -\sum(y\_i \times \log(p\_i)) \qquad [1]$$

Ecuación 1: Fórmula entropía cruzada

Donde:

- CCE representa la entropía categórica
- y_i es la variable objetivo binarizada para la clase i (1 si pertenece a la clase, 0 en caso contrario)
- p_i es la probabilidad predicha por el modelo para la clase i

Se explica la ecuación 1, con un ejemplo:

Se supone que hay una clasificación con 3 clases: calabacín pequeño, calabacín mediano y calabacín grandes. Para cada instancia de datos, se desea asignar una probabilidad a cada una de estas clases. Se tiene el conjunto de datos de entrenamiento y, para una instancia en particular, las probabilidades predichas por el modelo son:

- p_pequeño= 0.6
- p_grande=0.3
- p_mediano=0.1

La instancia de datos pertenece a la clase “pequeño”. Para representar esto, en forma binaria, la variable objetivo (y_i) para esta instancia es [1,0,0]. En este caso, la fórmula de entropía cruzada se calcula de la siguiente manera:

CEE= -[(1 * log(0.6)) + (0 * log(0.3)) + (0 * log(0.1))] = -(log(0.6)) ≈ 0.5108 [2]

En este ejemplo, la entropía cruzada categórica para esta instancia en particular sería aproximadamente de 0.5108

5.1.4. Algoritmo de Forward Propagation y Back Propagation

En términos generales, la Forward Propagation se enfoca en predecir imágenes de entrada y calcular la función de error, mientras que el Back Propagation actualiza las neuronas y sus pesos utilizando derivadas parciales en función de coste.

Para aplicar el algoritmo de back propagation, se requiere el gradiente, que se combina con regresión lineal. Sin embargo, en redes neuronales, el costo de la función cambia cuando se modifica un parámetro de la red, lo que se resuelve mediante el cálculo de la derivada parcial de la función de coste respecto a cada variable. Este proceso se complica debido a la gran cantidad de conexiones con otras neuronas de la misma capa y de capa anteriores.

El algoritmo de back propagation calcula el valor de gradiente dentro de la red neuronal, lo que permite trabajar de manera eficiente mediante la distribución del error hacia atrás y la modificación de los parámetros de manera recursiva capa por capa. Este proceso evita la necesidad de realizar múltiples combinaciones hacia adelante, lo que hace que el entrenamiento y la gestión de la red sean menos complicados. El algoritmo utiliza el descenso de gradiente para minimizar el costo de la red al calcular las derivadas parciales del vector gradiente [31] [32].

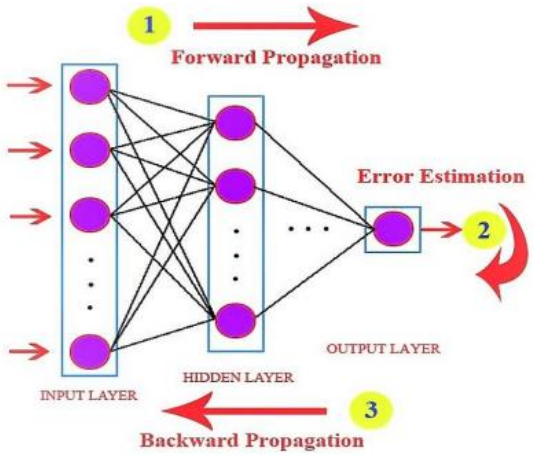


Figura 12: Algoritmo de forward propagation y back propagation [33]

Como se ha indicado previamente, las neuronas de una red se organizan en tres tipos de capas: la de capa de entrada (input layer), las capas ocultas (hidden layer) y las de capa de salida (output layer). En la misma capa se pueden colocar varias neuronas que recibirán la misma entrada de la capa anterior y producirán la misma salida para la siguiente capa.

Si las neuronas se colocan en secuencia, cada una procesará la información que la anterior ha generado, siguiendo un comportamiento jerárquico. Durante el proceso de DL, se conectan muchas neuronas en secuencia a través de capas ocultas, y cada una de ellas realiza una regresión lineal. Sin embargo, matemáticamente, sumar muchas líneas rectas conduce a otra línea recta, que se puede obtener mediante una única neurona u operación. Para evitar esta redundancia, se utilizan distorsiones o no linealidades en las rectas, a través de las funciones de activación.

5.1.5. Funciones de activación

Las funciones de activación se caracterizan por tener una curvatura o distorsión en su trazado, lo que le confiere un grado mayor a uno. En una neurona, en lugar de obtener solo una suma ponderada como salida a partir de los datos de entrada, esta suma pasa a ser el argumento de una función de activación que introduce deformaciones no lineales. El mapeo entre las entradas y salidas depende de la función de activación elegida, que se divide en dos subgrupos [34]:

- Funciones acotadas solo activan la neurona si el valor de entrada se encuentra dentro de un rango de valores determinado
- Funciones no acotadas no establecen límites en los valores para activar la neurona.

En el diseño de una red neuronal, es importante especificar los parámetros de activación y seleccionar previamente, según el tipo de datos y finalidad, una de las posibles funciones de activación disponibles.

La distorsión se aplica al final del cálculo interno de la neurona.

Entre las funciones de activación más comunes se encuentran: linear, sigmoide, tangente hiperbólica, unidad rectificada lineal (conocida como ReLU) y softmax.

Se procede a explicar las usadas en este trabajo:

- Función unidad rectificada lineal (conocida como ReLU), se corresponde con una función lineal cuando es positiva y un valor de cero constante cuando el valor es negativo. Al aplicar la función ReLU a la salida de cada neurona en la capa convolucional, se activan solo las neuronas que producen una respuesta positiva, y las demás se desactivan. Esto ayuda a reducir la complejidad computacional de la red y a mejorar la capacidad de generalización.

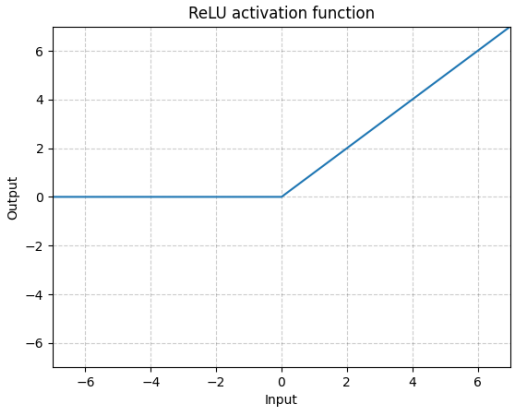


Figura 13: Función activación ReLU [35]

- Softmax, la cual se usa en la capa de salida de una red neuronal. La función toma un vector de entrada de números reales y lo transforma en un vector de la misma dimensión, pero con valores que están normalizados y suman 1, es decir, cada elemento de la salida de la capa Softmax representa la probabilidad de que la entrada dada pertenezca a una de las clases de salida. Por tanto, la probabilidad más alta es la que se elige como la predicción final.

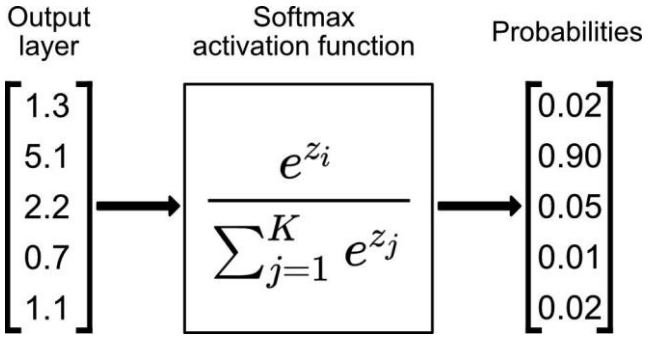


Figura 14: Función Softmax [36]

5.2. Etapas de la Red Neuronal Artificial

Una ANN está compuesta por dos etapas principales: la fase de entrenamiento y la fase de predicción. Se detallan a continuación:

5.2.1. Fase de entrenamiento

La primera etapa del proceso de entrenamiento de una red neuronal tiene como objetivo fundamental ajustar los parámetros de la red para obtener la solución óptima y minimizar el error en su función requerida. Este proceso se divide en tres tareas: el diseño de la estructura de a la red, el entrenamiento de la red y la validación.

- El diseño de la estructura de la red implica tomar varias decisiones, como la elección del conjunto de datos, el número de capas y neuronas en cada capa, y las funciones de activación a utilizar. Este proceso consiste en definir la topología de la red para lograr una buena eficiencia, lo cual puede ser difícil debido a que no existen reglas establecidas. En la práctica, se entrena la red con diferentes arquitecturas y configuraciones, midiendo su error para seleccionar finalmente la configuración más satisfactoria para el usuario.
- El entrenamiento en si es la modificación de los pesos de la red a través de la retropropagación, con el fin de que la red aprenda de manera similar a como lo hace el cerebro humano [37].
- La fase de validación se usa junto con la de entrenamiento para evitar caer en el sobreentrenamiento o subentrenamiento.
    - El sobreentrenamiento o overfitting es fácil de detectar cuando la precisión de entrenamiento alcanza la perfección mientras que la precisión de validación queda notablemente por debajo. En otras palabras, el modelo se ajusta demasiado a los datos de entrenamiento y no es capaz de generalizar para nuevos datos. Esto se debe a que el modelo está modelando el ruido de los datos de entrenamiento en lugar de la verdadera relación [38].

Para evitarlo, se puede aumentar la cantidad de datos o cambiar la estructura del modelo, prestando atención al número de iteraciones.

- El subentrenamiento o underfitting ocurre cuando el modelo no tiene la suficiente flexibilidad para ajustarse a la complejidad de los datos [39].

Para evitar que un modelo sobreentrene o subentrene los datos de entrenamiento, se definen los hiperparámetros. Los hiperparámetros son configurados antes de empezar el proceso de entrenamiento, y los usados en este proyecto son:

- Dropout que consiste en "apagar" algunas neuronas de las capas ocultas durante el entrenamiento, lo que no contribuyen a la propagación hacia adelante ni a la propagación hacia atrás del error. Esto previene que las neuronas se especialicen demasiado en el aprendizaje de ciertos patrones de entrenamiento [40] [41] [42].
- Número de épocas (epoch): Durante una época, el modelo procesa todos los datos de entrenamiento y ajusta sus parámetros de acuerdo con la función de pérdida que se está minimizando. Después de una época, el modelo puede actualizar sus parámetros o continuar entrenando para la siguiente época [43].
  Demasiadas épocas pueden llevar a sobreentrenamiento, mientras que muy pocas al subentrenamiento. La cantidad adecuada de épocas depende del modelo, el conjunto de datos.

5.2.2. Fase de predicción

La etapa final de un sistema de predicción consiste en ejecutar el modelo previamente configurado y entrenado con el fin de resolver el mismo problema, pero esta vez con entradas nunca vistas. Si el modelo está correctamente entrenado, debería haber aprendido y generalizado el método, en lugar de simplemente memorizar los datos de entrenamiento. Sin embargo, la calidad del conjunto de datos utilizado en el entrenamiento también es crucial para lograr un buen rendimiento en la predicción.

Es importante que el conjunto de datos sea amplio y variado, pero sin incluir patrones innecesarios que distraigan la atención del objetivo principal. Además, los datos deben contener todos los casos que cubre el modelo, sin tener que invertir una gran cantidad de recursos computacionales y tiempo en aprender conceptos irrelevantes [44].

La siguiente figura, muestra la forma de actuar de ambas fases:

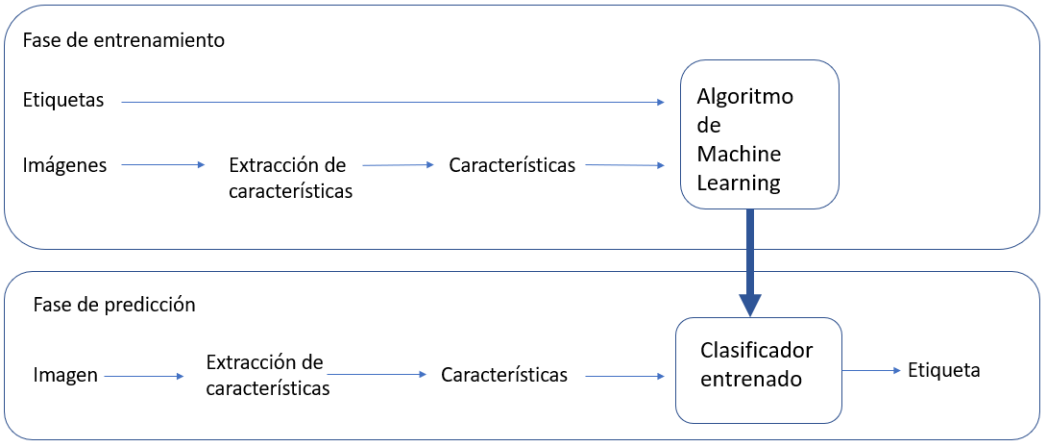


Figura 15: Fase de entrenamiento y fase de predicción

5.3. Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNNs), son un tipo específico de redes neuronales artificiales diseñadas para detectar patrones complejos en grandes conjuntos de datos, destacando en aplicaciones de procesamiento de imágenes y reconocimiento de voz. Las CNN simulan la capacidad humana de reconocimiento con una complejidad inalcanzable para otros tipos de ANN. En este contexto, se puede enfocar la explicación desde la perspectiva del proyecto de clasificación de imágenes.

Las CNNs funcionan mediante la aplicación de una serie de filtros o kernels sobre la imagen de entrada, tratando cada píxel como una variable independiente. Estos filtros son matrices de números que se desplazan por la imagen para extraer características como bordes, formas y patrones. La salida de cada filtro se convierte en una nueva imagen, que se utiliza como entrada para el siguiente filtro. Este proceso se repite varias veces, con múltiples filtros, hasta que se extraen características cada vez más complejas. Este proceso se denominará proceso convolucional.

Después de la fase de extracción de características, la CNN utiliza una o más capas densas (completamente conectadas) para clasificar la imagen. Estas capas densas toman como entrada las características extraídas y las combinan para producir la salida final de la red, que es la clasificación de la imagen. Esta parte se corresponde con la parte de clasificación.

Se detalla a continuación:

5.3.1. Parte Convolucional

La convolución es un método matemático que se usa para el tratamiento y el reconocimiento de imágenes.

Este proceso se compone de tomar grupo de píxeles cercanos de la imagen de entrada y operar matemáticamente (producto escalar) con una pequeña matriz denominada filtro o kernel. A medida que se desplaza el kernel, se va obteniendo una nueva imagen filtrada por este. Estas

nuevas imágenes lo que están dibujando son ciertas particularidades de la imagen original, lo que facilitará a poder distinguir un objeto de otro [45].

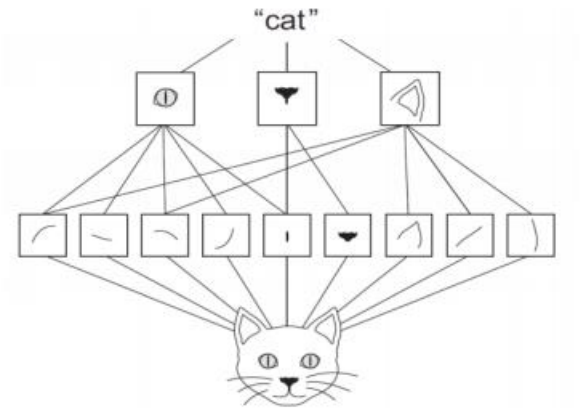


Figura 16: Ejemplo de filtro o kernel

Si se realiza una convolución adicional a partir de esa nueva imagen, el número de neuronas en la siguiente capa aumentaría significativamente, lo que se traduce en una carga computacional más alta. Para evitar esto, se utiliza el proceso de submuestreo en el que se minimiza el tamaño de las imágenes filtradas mientras se mantienen las particularidades más importantes detectadas por cada filtro. Existen varios métodos de submuestreo, el usado en este proyecto ha sido el max-pooling:

- Método de submuestreo de max-pooling: es una operación utilizada en CNN para reducir el tamaño espacial de la salida de la capa convolucional. Consiste en dividir la imagen en regiones no solapantes y para cada región seleccionar el valor máximo dentro de esa región, ignorando los demás.
  De esta manera, se reduce la cantidad de información de la imagen, disminuyendo el número de parámetros y acelerando el proceso de entrenamiento de la red.
  Además, el max-pooling también ayuda a evitar el sobreajuste, al reducir la sensibilidad de la red a pequeñas variaciones en la posición de los objetos en la imagen.
  En la siguiente imagen puede verse un ejemplo, a la izquierda una matriz de 4x4, que representa una entrada inicial, se le aplica un filtro de 2x2. Para cada región barrida por el filtro, el Max-Pooling tomará el máximo [45] [46] [47].

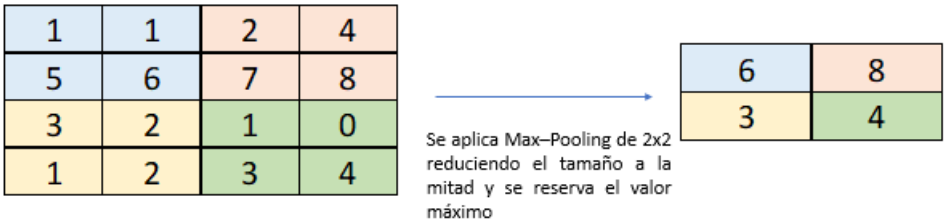


Figura 17: Max-Pooling

La primera convolución puede encontrar características primitivas como líneas o curvas. A medida que se empleen más capas con las convoluciones, los mapas de características serán capaces de reconocer formas más complejas, y el conjunto total de capas de convoluciones podrá "ver".

5.3.2. Parte de clasificación

La parte de clasificación de una CNN se encarga de utilizar las características extraídas por la parte convolucional y clasificar la imagen en una o varias categorías. Esta parte consiste en una serie de capas totalmente conectadas también conocidas como perceptrón multicapa (*Multi Layers Perceptron*) [48].

La entrada de la parte de clasificación es el código CNN obtenido en la salida de la parte convolucional, que contiene información sobre las características relevantes de la imagen. Este código se ejecuta a través de una serie de capas totalmente conectadas, en las que cada neurona está relacionada con todas las neuronas de la capa anterior. Cada una de estas neuronas aplica una función de activación no lineal a la combinación lineal de sus entradas para producir su salida.

La última capa de la parte de clasificación es una capa de salida que produce las probabilidades de la imagen que pertenece a cada una de las clases posibles. La función de activación utilizada en la capa de salida suele ser la función softmax, que garantiza que las probabilidades sumen 1.

Durante el entrenamiento, se utiliza el algoritmo de optimización para ajustar los pesos y los sesgos (bias) de las neuronas de la parte de clasificación con el objetivo de minimizar la función de perdida, que mide la diferencia entre las probabilidades de salida de la red y las etiquetas de las imágenes de entrenamiento.

Una vez que la red está entrenada, se puede usar para clasificar nuevas imágenes en una o varias categorías.

6. Software de reconocimiento por imágenes.

En este apartado, se llevará a cabo una detallada descripción de las herramientas y el entorno que se ha seleccionado para familiarizarse con las redes neuronales. Se presentará el conjunto de datos que se ha utilizado en el proyecto, así como sus características. Además, se proporcionará el código que se ha desarrollado para simular el proyecto y se ofrecerán algunas definiciones para entender mejor los parámetros internos de esta red neuronal.

6.1. Entorno

En esta sección, se presenta el software utilizado junto con una breve explicación de su funcionamiento y ventajas.

El software utilizado es Anaconda, un software gratuito y multiplataforma que proporciona una amplia variedad de herramientas. Se utilizó Anaconda debido a que permite acceder a diferentes entornos de programación en lenguajes populares como Python o R. Estos entornos se conocen como entornos de desarrollo integrado (IDE), y son aplicaciones que facilitan el desarrollo de código [49].

Figura 18: Logo de Anaconda

En este proyecto, se utilizó Spyder como entorno de desarrollo integrado. Spyder es un software gratuito y de código abierto que ofrece una amplia gana de herramientas para la codificación de scripts en Python, como la inspección y depuración de código.

Una de las ventajas de trabajar con Spyder es su editor de código, que permite escribir y ejecutar código de manera interactiva. Además, cuenta con herramientas que facilitan la identificación de errores y la corrección de estos, lo que acelera el proceso de desarrollo.

Otra ventaja de Spyder es su integración con otras herramientas y bibliotecas utilizadas en el campo de la programación, como NumPy, Matplolib y Pandas. Esto permite trabajar de manera más eficiente y productiva en proyectos que requieren el uso de estas herramientas.

Figura 19: Logo de Spyder

El lenguaje de programación elegido para implementar el código ha sido Python 3.6, ya que es compatible con las librerías utilizadas en el proyecto y ofrece todas las capacidades necesarias

para el mismo. Python es un lenguaje interpretado y orientado a objetos, lo que lo hace fácil de aprender y suficientemente versátil como para abordar una amplia gama de tareas. Además, es un lenguaje de código abierto y su popularidad ha crecido enormemente desde su lanzamiento en 1991. Esto se debe en parte a su compatibilidad con varios sistemas operativos (Linux, Windows, Mac) y al hecho de que no requiere de equipos de alta capacidad para su programación. En este proyecto, Python ha sido utilizado en conjunto con otras herramientas y librerías, como Spyder y Anaconda, que proporciona un entorno completo y eficiente para la programación [50].

Figura 20: Logo de Python

En este trabajo, se exprimen numpy como recurso de la siguiente interfaz de programación de aplicaciones (API).

Se introducen a continuación, las librerías usadas:

- Glob: es una librería para buscar archivos que coinciden con un patrón en un directorio. Se ha usado para cargar imágenes y datos de diferentes carpetas y subcarpetas.
- Random as rn: esta librería se ha usado para generar valores aleatorios para inicializar los pesos de la red neuronal, y para aleatorizar la selección de datos de entrenamiento y prueba.
- Numpy as np: proporciona herramientas para trabajar con matrices y vectores, lo que resulta muy útil en procesamiento de imágenes y en la implementación de redes neuronales. Se utilizada para manipular y procesar datos de entrada.
- Pandas as pd: librería para el análisis de datos en Python. Proporciona estructuras de datos para trabajar con datos tabulares y series temporales. Se utiliza para cargar y procesar datos tabulares de entrada o para exportar resultados.
- Matplotlib.pyplot as plt: librería para visualizar datos basada en matplotlib. Proporciona herramientas para crear gráficos estadísticos.
- Cv2: es una librería de visión por computadora de código abierto. Facilita herramientas para el procesamiento de imágenes y vídeos. Se utiliza para realizar operaciones de preprocesamiento de imágenes, antes de que sean utilizadas como entrada en la red neuronal.

Se han usado dos *frameworks* específicos: Keras y TensorFlow, ambas son intuitivas, con sintaxis e interfaces simples [51].

La biblioteca TensorFlow proporciona un software gratuito de código abierto desde noviembre de 2015. A través de ella, se puede ejecutar otro framework de alto nivel llamado Keras, que se encarga de realizar tareas de aprendizaje en Python. Además, ambas plataformas combinan perfectamente y se utilizan para construir y entrenar redes neuronales ya que permiten identificar patrones similares a los cognitivos.

Por último, la librería "sklearn.metrics" proporciona una variedad de herramientas para evaluar la precisión y el rendimiento de los modelos de aprendizaje automático, incluyendo la clasificación y la matriz de confusión.

Se introduce así, la matriz de confusión, como una herramienta utilizada para evaluar el modelo de clasificación. La matriz de confusión muestra la cantidad de muestras que han sido clasificadas correctamente, y las que lo han sido incorrectamente. En este proyecto, determina que calabacines han sido clasificados correctamente en términos de su calibre, y cuales han sido clasificados incorrectamente.

Se ha usado también, Google Drive para almacenar la base de datos y permitir una comunicación fluido a Google Colab. Colab se ha usado para desarrollar y entrenar las arquitecturas de las redes ya que es gratuito y cuenta con recursos de ejecución suficientes para tareas de DL, como acceso a servidores con GPU, procesadores y memoria RAM.

```
import glob
import random as rn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop, Adam
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
```

Código 1: Librerías importadas

6.2. Base de datos

El conjunto de datos utilizado para el entrenamiento de la red neuronal fue generado de manera manual, ya que no se encontró ninguna base de datos disponible que proporcionara una selección de calabacines clasificados por su calibre. Para crear esta base de datos, se estableció una altura específica para tomar las imágenes y se clasificaron según su tamaño.

En la tabla que se muestra a continuación, queda resumida la estructura que siguen los datos. Se han separado en tres directorios diferentes, cuyos nombres se corresponden con las categorías de las imágenes para hacer así el etiquetado.

Los tres calibres por longitud con los que se ha trabajado son los establecidos según el Reglamento (CEE) nº 1292/81:

- Pequeño, de 7 cm a 14 inclusive, se le asignará la etiqueta de 0.
- Mediano, de 14 cm exclusive a 21 cm inclusive, se le asignará la etiqueta de 1.
- Grande, de 21 cm exclusive a 30cm, se le asignará la etiqueta de 2.

	Entrenamiento (Train)	Validación (Val)	Prueba (Test)
Grandes	38	2	7
Medianos	33	2	6
Pequeños	35	2	7

Tabla 1: Volumen de la base de datos

Finalmente, la base de datos está compuesta por 132 fotos, de las cuales 136 han sido para usadas para el entrenamiento, 6 para validación y 20 para prueba. La separación está en torno al 80% para entrenamiento, 5% para validación y 15% para prueba.

En las siguientes figuras, se muestran ejemplos de las carpetas de entrenamiento de los calibres a diferenciar, las dimensiones de los píxeles son aproximadamente 242x432.

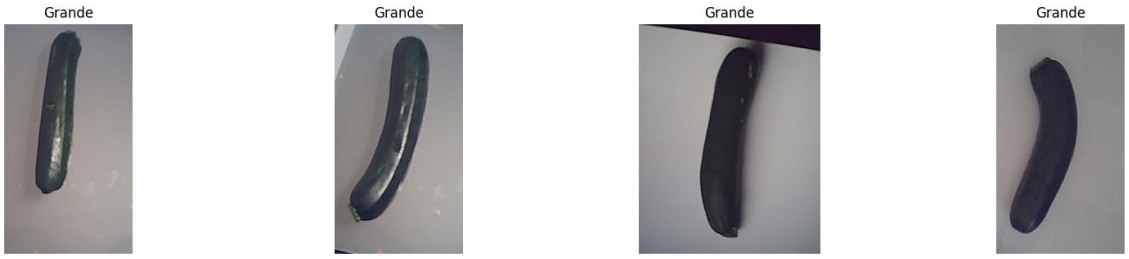


Figura 21: Muestras de entrenamiento categoría 'Grande'

Figura 22: Muestras de entrenamiento categoría 'Mediano'

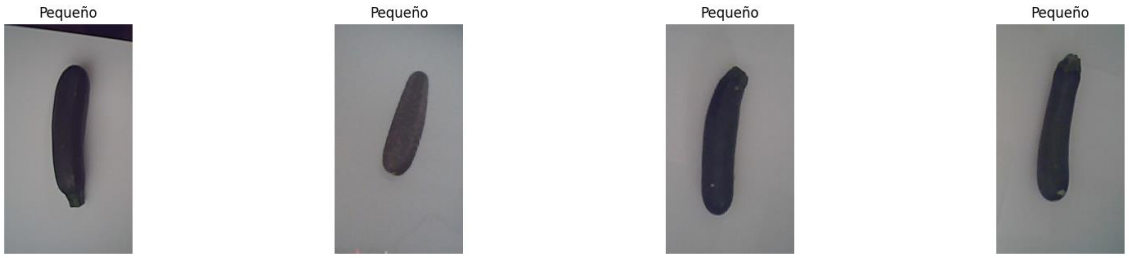


Figura 23: Muestras de entrenamiento categoría 'Pequeño'

La metodología aplicada para el desarrollo de la CNN fue equivalente al empleado en otras arquitecturas para BBDD (base de datos) más grandes, con la salvedad de que se debía prestar especial atención en el tamaño y ajuste de los hiperparámetros, dado que los resultados presentaban cierto sesgo, traduciéndose en un proceso de experimentación largo y meticuloso.

6.3. Implementación

Tras exponer los fundamentos teóricos del amplio universo del DL y, presentar el conjunto de datos utilizado para entrenar la red, así como el entorno de desarrollo empleado, se va a abordar los archivos de código escritos en Python para implementar la CNN.

6.3.1. Estructura de directorios

Para entender como el sistema procesa los datos, es importante tener una comprensión clara de cómo se ha organizado la información. En este proyecto, hay tres subcarpetas principales: entrenamiento (test), validación (val) y prueba (test). Cada carpeta contiene tres subcarpetas, que representan las diferentes categorías de la clasificación: grandes, medianos, pequeños.

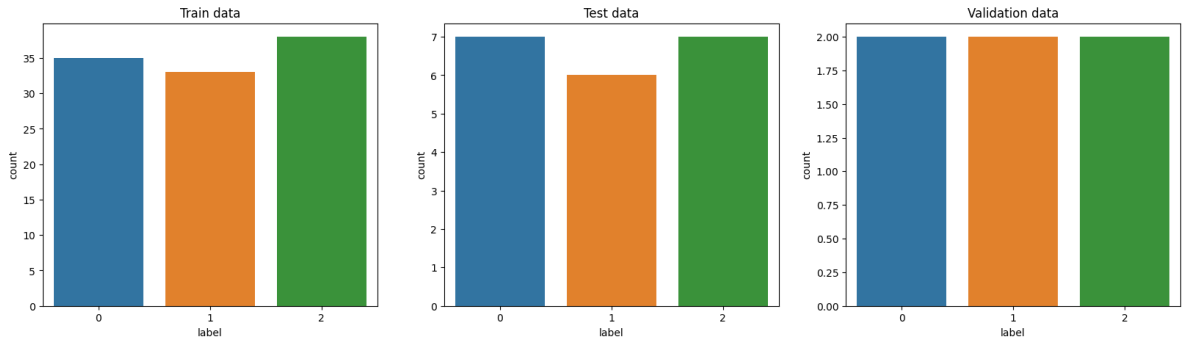


Figura 24: Histograma con la composición de cada una de las divisiones que componen la base de datos

En el siguiente código, se muestra cómo se han definido los directorios.

```
path = 'Calabacines/'


# Definición de directorios
train_pequeño_dir = path + 'train/Pequeños/'
train_mediano_dir = path + 'train/Medianos/'
train_grande_dir = path + 'train/Grandes/'

test_pequeño_dir = path + 'test/Pequeños/'
test_mediano_dir = path + 'test/Medianos/'
test_grande_dir = path + 'test/Grandes/'


val_pequeño_dir = path + 'val/Pequeños/'
val_mediano_dir = path + 'val/Medianos/'
val_grande_dir = path + 'val/Grandes/'
```

Código 2: Definición de los directorios

6.3.2. Preprocesado de imágenes: Data Augmentation

El Data Augmentation es una técnica muy utilizada en el DL, consiste en aumentar artificialmente el tamaño del conjunto de datos existente mediante la generación de nuevas imágenes a partir de las imágenes originales [52].

Esta técnica es muy útil cuando se dispone de un conjunto de datos pequeños, como es el caso, ya que permite aumentar la variabilidad de los datos, y, por tanto, mejorar la capacidad del modelo para generalizar nuevas imágenes que no están presentes en el conjunto de datos de entrenamiento [53].

Se han usado las siguientes trasformaciones:

- Rotation_range: Rota la imagen hasta 10 grados en cualquier dirección.
- Zoom_range: Amplia o reduce la imagen hasta un máximo del 10%.
- Width_shift_range: Desplaza la imagen horizontalmente hasta un 10% del ancho.
- Height_shift_range: Desplaza la imagen verticalmente hasta un 10% de la altura.

En la siguiente figura, puede verse un ejemplo del Data Augmentation, donde la imagen de la izquierda es la imagen original, y las imágenes de la derecha a las que se le ha aplicado las transformaciones:

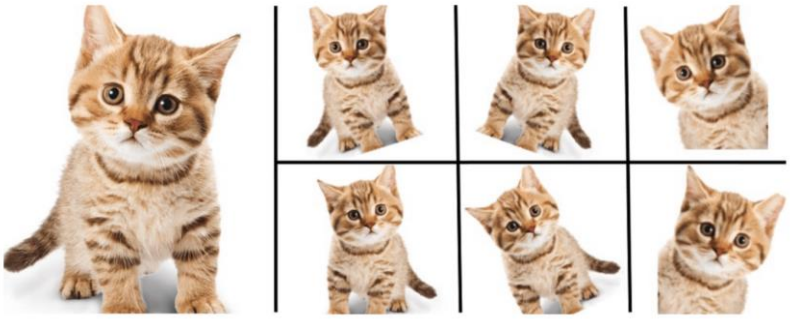


Figura 25: Ejemplo de Data Augmentation [54]

Se presenta el preprocesado de imágenes en el código:

```
# Generador de imágenes
datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range = 0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False,
    vertical_flip=False
)

datagen.fit(X_train)#solo se aplica en el training set
```

Código 3: Preprocesado de imágenes con ImageDataGenerator

6.3.3. Entrenamiento

En este subapartado, se va a completar el diseño de la CNN, de acuerdo con los conceptos explicados en anteriores capítulos:

Se muestra el código:

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu', input_shape=(196, 196, 1)))
#model.add(Conv2D(filters=8, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3)))


model.add(Conv2D(filters=16, kernel_size=(2,2), padding='same', activation='relu'))
#model.add(Conv2D(filters=16, kernel_size=(2,2), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(3, activation='softmax'))

optimizer = Adam(learning_rate=0.0001)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

callback = EarlyStopping(monitor='loss', patience=7)
history = model.fit(datagen.flow(X_train,y_train, batch_size=2), validation_data=(X_test, y_test), epochs = 100, callbacks=callback, verbose = 1)
```

Código 4: Entrenamiento de la CNN

Primero, se utiliza la capa Conv2D para extraer características de la imagen de entrada. La capa Conv2D tiene 32 filtros y utiliza un tamaño de kernel de (3,3) con activación de ReLU y el mismo relleno (padding = same). En este caso, se utiliza el valor 'same', que indica que se debe agregar relleno a la imagen para que la salida de la convolución tenga las mismas dimensiones de entrada.

A continuación, se utiliza la capa MaxPooling2D para reducir la dimensionalidad de la salida de la capa convolucional anterior mediante el submuestreo. La capa MaxPooling2D tiene un tamaño de ventana de (3,3).

En el tercer párrafo, se agrega otra capa Conv2D con 16 filtros y un tamaño de kernel de (2,2), también con activación de ReLU y el mismo relleno. Se utiliza nuevamente una capa de MaxPooling2D con tamaño de ventana (2,2) para reducir la dimensionalidad.

A continuación, se utiliza la capa Flatten para convertir la salida de la capa anterior en un vector unidimensional.

En el quinto párrafo, se utiliza una capa Dense de 32 neuronas con activación ReLU, seguida de una capa Dropout con 20% de probabilidad para evitar el sobreentrenamiento. Se utiliza una capa Dense de salida con 3 neuronas, y activación softmax, que clasifica las imágenes en una de las tres categorías posibles: "pequeños", "medianos" y "grandes".

El modelo se ejecuta con la función de perdida de entropía cruzada categórica y el optimizador Adam con una tasa de aprendizaje de 0.0001, que se usará para ajustar los pesos del modelo durante el entrenamiento. En este caso, se utiliza la pérdida de entropía cruzada categórica (`categorical_crossentropy´) como función de pérdida y se registra la precisión (`accuracy´) como métrica.

Además, se utiliza EarlyStopping como callback para detener el entrenamiento si la pérdida no mejora después de 7 épocas. El modelo se entrena durante 100 épocas.

Se ajusta el modelo a los datos de entrenamiento (`model.fit(…)´) utilizando un generador de datos (`datagen.flow (…)´) para el aumento de datos. Se especifica también el conjunto de validación (`validation_data´), el número de épocas y los callbacks a utilizar durante el entrenamiento.

Se presenta el resumen de la arquitectura:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 196, 196, 32)      320

 max_pooling2d_4 (MaxPooling  (None, 65, 65, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 65, 65, 16)        2064

 max_pooling2d_5 (MaxPooling  (None, 32, 32, 16)       0
 2D)

 flatten_2 (Flatten)         (None, 16384)             0

 dense_4 (Dense)             (None, 32)                524320

 dropout_2 (Dropout)         (None, 32)                0

 dense_5 (Dense)             (None, 3)                 99

=================================================================
Total params: 526,803
Trainable params: 526,803
Non-trainable params: 0
_________________________________________________________________
```

Código 5: Resumen de la arquitectura

Para terminar este subcapítulo, se guarda el modelo en un archivo HDF5. Los archivos HDF5 se utilizan para almacenar datos entrenados donde se guarda tanto la arquitectura del modelo como los pesos y los valores de los parámetros aprendidos durante el entrenamiento, para evitar entrenar al sistema de nuevo en cada predicción y solo tener que reusarlo.

```
[ ]  !ls

     Calabacines  gdrive  my_model.h5  sample_data

[ ]  # Save the entire model to a HDF5 file.
     # The '.h5' extension indicates that the model should be saved to HDF5.
     model.save('my_model.h5')
```

Código 6: Guardado del modelo y los pesos de la red final

6.4. Evaluación de los resultados

En este subcapítulo se van a estudiar los resultados obtenidos.

Tras ejecutarse el código, se muestra en las siguientes dos capturas de pantallas la evolución de las diez primeras épocas y de las diez ultimas épocas:

```
Epoch 1/100
53/53 [==============================] - 5s 72ms/step - loss: 1.1191 - accuracy: 0.2830 - val_loss: 1.0955 - val_accuracy: 0.3500
Epoch 2/100
53/53 [==============================] - 4s 75ms/step - loss: 1.1013 - accuracy: 0.3113 - val_loss: 1.0832 - val_accuracy: 0.3500
Epoch 3/100
53/53 [==============================] - 3s 63ms/step - loss: 1.1094 - accuracy: 0.3019 - val_loss: 1.0782 - val_accuracy: 0.7000
Epoch 4/100
53/53 [==============================] - 5s 86ms/step - loss: 1.1014 - accuracy: 0.3396 - val_loss: 1.0768 - val_accuracy: 0.5500
Epoch 5/100
53/53 [==============================] - 4s 67ms/step - loss: 1.0921 - accuracy: 0.3585 - val_loss: 1.0682 - val_accuracy: 0.3500
Epoch 6/100
53/53 [==============================] - 3s 63ms/step - loss: 1.0925 - accuracy: 0.3585 - val_loss: 1.0683 - val_accuracy: 0.3500
Epoch 7/100
53/53 [==============================] - 3s 64ms/step - loss: 1.0880 - accuracy: 0.3679 - val_loss: 1.0601 - val_accuracy: 0.3500
Epoch 8/100
53/53 [==============================] - 5s 100ms/step - loss: 1.0845 - accuracy: 0.4245 - val_loss: 1.0516 - val_accuracy: 0.3500
Epoch 9/100
53/53 [==============================] - 3s 61ms/step - loss: 1.0792 - accuracy: 0.3774 - val_loss: 1.0444 - val_accuracy: 0.3500
Epoch 10/100
53/53 [==============================] - 5s 89ms/step - loss: 1.0764 - accuracy: 0.3962 - val_loss: 1.0282 - val_accuracy: 0.4500
```

Código 7: 10 primeras Epochs

```
Epoch 79/100
53/53 [==============================] - 5s 99ms/step - loss: 0.6597 - accuracy: 0.6981 - val_loss: 0.4632 - val_accuracy: 0.8500
Epoch 80/100
53/53 [==============================] - 3s 65ms/step - loss: 0.7378 - accuracy: 0.6792 - val_loss: 0.4390 - val_accuracy: 0.8500
Epoch 81/100
53/53 [==============================] - 5s 101ms/step - loss: 0.6795 - accuracy: 0.6887 - val_loss: 0.3460 - val_accuracy: 0.9500
Epoch 82/100
53/53 [==============================] - 3s 62ms/step - loss: 0.6150 - accuracy: 0.7358 - val_loss: 0.4005 - val_accuracy: 0.9000
Epoch 83/100
53/53 [==============================] - 5s 95ms/step - loss: 0.6993 - accuracy: 0.7075 - val_loss: 0.5390 - val_accuracy: 0.8000
Epoch 84/100
53/53 [==============================] - 3s 62ms/step - loss: 0.7423 - accuracy: 0.6604 - val_loss: 0.4078 - val_accuracy: 0.9000
Epoch 85/100
53/53 [==============================] - 3s 62ms/step - loss: 0.6756 - accuracy: 0.7264 - val_loss: 0.4172 - val_accuracy: 0.8500
Epoch 86/100
53/53 [==============================] - 5s 86ms/step - loss: 0.7350 - accuracy: 0.6226 - val_loss: 0.4331 - val_accuracy: 0.9000
Epoch 87/100
53/53 [==============================] - 3s 63ms/step - loss: 0.7533 - accuracy: 0.7075 - val_loss: 0.3812 - val_accuracy: 0.9500
Epoch 88/100
53/53 [==============================] - 3s 65ms/step - loss: 0.6590 - accuracy: 0.7075 - val_loss: 0.3794 - val_accuracy: 0.9500
Epoch 89/100
53/53 [==============================] - 5s 92ms/step - loss: 0.6729 - accuracy: 0.6792 - val_loss: 0.3526 - val_accuracy: 0.9500
```

Código 8: 10 últimas Epochs

Dado que la pérdida no ha mejorado durante 7 epochs, se detuvo el entrenamiento.

Se presenta a continuación, la curva de pérdidas y la curva de accuracy. La curva de pérdidas (izquierda) muestra cómo evoluciona la función de pérdidas a lo largo de las epochs del modelo, mientras que la curva de accuracy muestra como varía la precisión del modelo en la clasificación correcta de los datos a lo largo de las epochs. Estas curvas muestras información importante sobre el rendimiento y la capacidad de generalización del modelo:

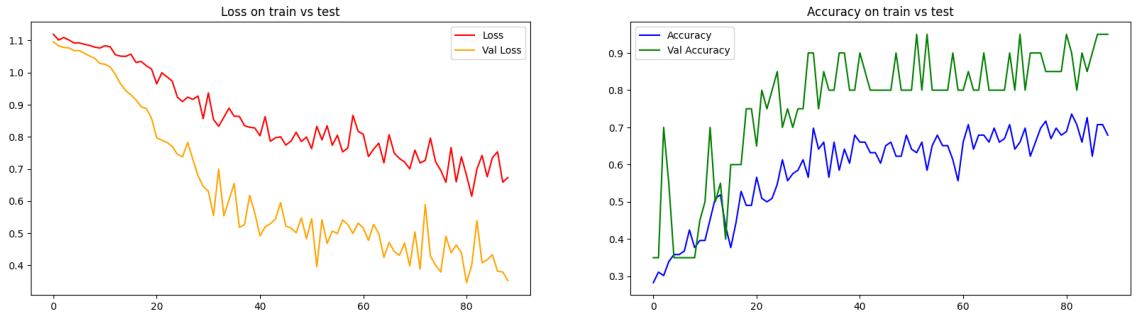


Figura 26: Izquierda, curva de pérdidas del subconjunto del training (rojo) y subconjunto de la prueba (amarillo).

Derecha, curva de accuracy del subconjunto training en azul, y en verde del subconjunto del test.

Se aprecia en la figura superior izquierda, existe una diferencia de aproximadamente 0.3 de valor de pérdida, en concordancia con los resultados del accuracy de la derecha, que son próximos del 15%.

La interpretación de estas métricas nos sugería que el modelo puede tener problemas durante el aprendizaje en la fase de entrenamiento conduciéndolo a una incapacidad de generalización para nuevos datos.

Durante el proceso de experimentación y diseño de la arquitectura se trataron de abordar distintos enfoques para analizar las patologías de la base de datos o de la CNN.

Uno de los primeros diagnósticos fue un diseño en complejidad progresiva de la CNN. Tanto con modelos más sencillos y complejos los resultados empeoraban significativamente. Paralelamente se llevó a cabo un proceso de ajuste de hiperparámetros.

Por lo que la CNN actual indica que tiene la capacidad expresiva suficiente para capturar los patrones de las imágenes de la base de datos. Descartando un posible problema de underfitting.

Otro punto donde se prestó especial atención fue durante el preprocesado de las imágenes en la que, además de hacer una estandarización clásica, se llevó a cabo una aumentación sintética en la que la alteración con respecto a las imágenes de partida era moderada o leve.

En consecuencia, con lo anterior, y considerando la previa presentación de la base de datos se concluye que el principal problema de las métricas obtenidas proviene del tamaño reducido de la base de datos y la calidad de este.

El siguiente código muestra los porcentajes:

```
score = model.evaluate(X_test, y_test, verbose = 0) #Se evalua cuántas imágenes de la subdivisión test es capaz de predecir bien (calculando las pérdidas y el accuracy)

print(f"Test Loss: {score[0]}")
print(f"Test Accuracy: {score[1] * 100:.2f}%")
```

```
Test Loss: 0.3526293635368347
Test Accuracy: 95.00%
```

```
[ ] score2 = model.evaluate(X_train, y_train, verbose = 0)  #Se evalua cuantas imagenes de la subdivisón train es capaz de predecir bien
print(f"Train Loss: {score2[0]}")
print(f"Train Accuracy: {score2[1] * 100:.2f}%")
```

```
Train Loss: 0.5105329751968384
Train Accuracy: 80.19%
```

Código 9: Porcentajes de pérdidas y accuracy

El siguiente fragmento de código va a generar la matriz de confusión. Una matriz de confusión se utiliza en problemas de clasificación para evaluar el rendimiento del modelo al predecir las clases de un conjunto de datos. La matriz muestra, por tanto, la cantidad de muestras que han sido clasificadas correcta e incorrectamente por cada clase.

```
conf_m = confusion_matrix(y_test, y_test_hat) #Se calcula la matriz de confusión
clas_r = classification_report(y_test, y_test_hat) #Se genera un informe de clasificación

plt.figure(figsize=(5,5))
sns.set(font_scale=1.2)
ax = sns.heatmap(conf_m, annot=True,xticklabels=['P','M', 'G'], yticklabels=['P','M', 'G'], cbar=False, cmap='Blues',linewidths=1, linecolor='black', fmt='.0f')
plt.yticks(rotation=0)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
ax.xaxis.set_ticks_position('top')
plt.title('Confusion matrix - test data\n(M - Mediano, P - Pequeño, G - Grande)')
plt.show()

print('Classification report on test data')
print(clas_r)
```

Código 10: Generación de matriz de confusión y un informe de clasificación

La siguiente tabla muestra la matriz de confusión, donde las filas simbolizan las clases reales y las columnas, las clases predichas. Cada celda de la matriz contiene el recuento de las muestras que pertenecen a una clase especifica y han sido clasificadas en otra clase según las predicciones del modelo.

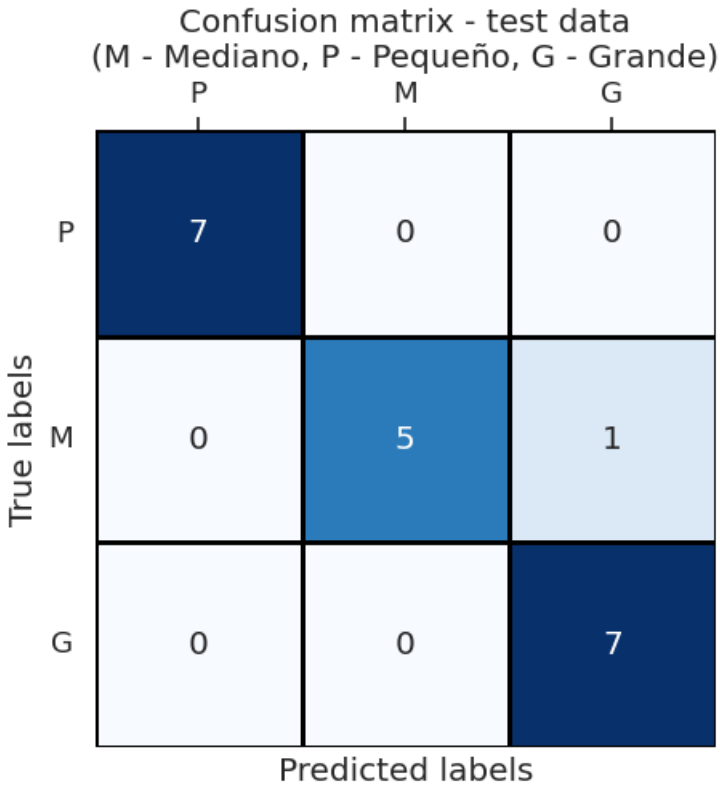


Tabla 2: Matriz de confusión

El modelo presenta un buen rendimiento, solo ha clasificado erróneamente un calabacín mediano como grande, se justifica con el 95% obtenido del Test Accuracy.

El informe de la clasificación se presenta a continuación:

Classification report on test data

	precision	recall	f1-score	support
0	1.00	1.00	1.00	7
1	1.00	0.83	0.91	6
2	0.88	1.00	0.93	7
accuracy			0.95	20
macro avg	0.96	0.94	0.95	20
weighted avg	0.96	0.95	0.95	20

Tabla 3: Informe de clasificación

La interpretación de los resultados es:

- Precisión: la presión re refiere a la proporción de predicciones positivas que fueron acertadas. En general, se observa una alta precisión en todas las clases, lo que indica que la mayoría de las predicciones son correctas.
- Recall: el recall (también conocido como sensibilidad o tasa de verdades positivos) se refiere a la proporción de instancias positivas que fueron correctamente identificadas. Se observa un alto recall en todas las clases, lo que indica que el modelo pudo identificar correctamente la mayoría de las instancias de cada clase.
- F1-score: el F1-score es una medida que combina tanto la precisión como el recall en un solo valor. Un valor de 1 indica un F1-score perfecto, mientras que un valor más bajo indica un equilibrio entre precisión y recall. En este caso, se observan valores altos de F1-score en todas las clases, lo que indica un buen equilibrio entre precisión y recall.
- Support: el soporte se refiere al número de instancias en cada clase en el conjunto de prueba.

El código completo puede verse en el Anexo – Códigos.

7. Presentación de los robots colaborativos

Una vez que se ha desarrollado una red neuronal convolucional (CNN) capaz de detectar el calibre de los calabacines, en esta sección se introduce los tipos de robots más usados en el mercado y el robot colaborativo para el que se ha el diseñado la pinza (o en inglés, gripper).

Este gripper será utilizado en conjunto con el robot para clasificar los calabacines en función del tamaño predicho por la CNN. De esta manera, se establecerá un sistema integral que combina la detección de calibre y la manipulación colaborativa para optimizar la clasificación de los calabacines.

El objetivo de diseñar un gripper específico para calabacines es proporcionar adaptabilidad precisa, eficiencia en el transporte, un ajuste preciso y eficaz en el agarre y transporte de estos, buscando agilizar los procesos de producción y envasado mediante el uso de un robot colaborativo.

7.1. Estado del arte

El estado del arte en la robótica colaborativa ha experimentado avances significativos en el contexto de la Industria 4.0 y las fábricas inteligentes. Esta revolución industrial implica la incorporación extendida de nuevas tecnologías en los procesos industriales, en este sentido, la robótica avanzada y colaborativa se refiere a robots industriales con elementos de sensorización que pueden ser integrados en entornos de fabricación ágil, trabajando en colaboración con los operarios y priorizando la seguridad del trabajador. Estos sistemas ciberfísicos (CPS) poseen la capacidad de utilizar información del mundo virtual, permitiendo el aprendizaje y la evolución a través de técnicas de IA, así como la interacción con objetos físicos.

Los sistemas CPS encuentran aplicación en diversos sectores, como la fabricación, energía, transporte, cuidades inteligentes y salud. Algunas soluciones aplicadas en el contexto de la Industria 4.0 incluyen el control de maquina en plantas de producción, la monitorización de máquinas para optimizar su operación y mantenimiento, la utilización de robots colaborativos que interactúan entre sí y con su entorno [55].

La robótica avanzada y colaborativa desempeña un papel destacado como habilitador tecnológico en la Industria 4.0, tanto en el presente, con empresas lideres en cada sector que comienzan a implementar esta tecnología, como en el futuro, donde se espera un crecimiento exponencial en su adaptación. Esta tecnología permite la hibridación del mundo físico y digital, enlazando el mundo físico con el virtual para evolucionar hacia una industria inteligente. En la figura adjunta, muestra un gráfico de las ventas de los robots industriales desde 2004 a 2020, con una previsión de hasta 2024.

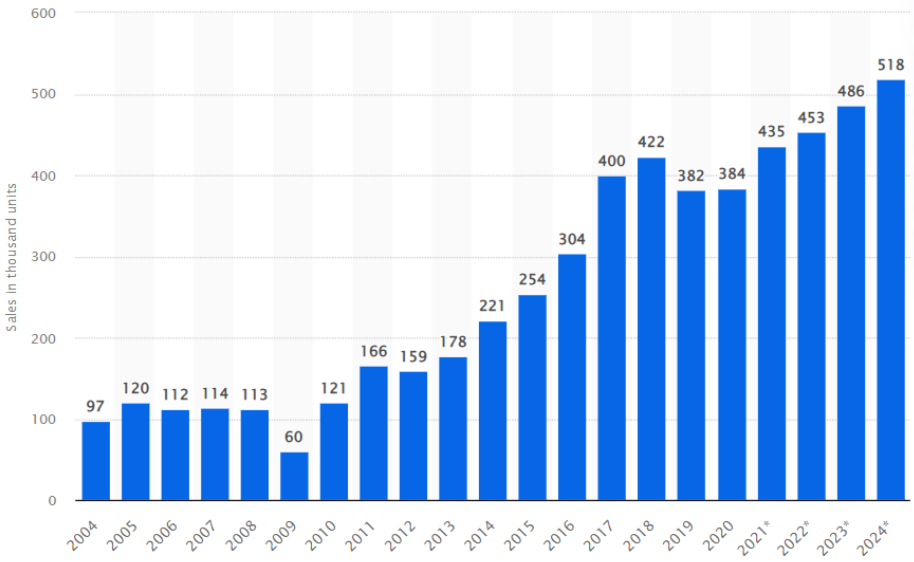


Figura 27: Ventas en miles de unidades por año de los robots industriales [56].

En la industria agroalimentaria y bio, las nuevas tecnologías, la mejora de la productividad y la personalización de productos demandada por los consumidores están impulsando la implementación de la automatización y robótica en el marco de la cuarta revolución industrial. El objetivo de esta automatización es cumplir con los altos estándares de calidad, productividad, eficiencia y seguridad alimentaria exigidos por el mercado y los consumidores.

La integración de diferentes niveles de automatización asegura la competitividad en la industria alimentaria, y los robots brindan ventajas al trabajar de manera confiable, precisa y rápida, incluso en condiciones de trabajo difíciles. Se aplican en áreas industriales como la producción, el embalaje, el almacenamiento y la logística. En la industria alimentaria, la automatización y robotización se emplean principalmente en la etapa final del proceso productivo, especialmente en el empaquetado, paquetizado y paletizado de productos terminados.

Algunos ejemplos de automatización y robotización en este sector serían los siguientes:

- El proyecto “Hands Free Hectare” fue concebida en la Universidad de Harper en Reino Unido, y representa la primera implementación automatizada de cultivo de trigo en el mundo.

Figura 28: "Hands Free Hectare [57]"

- Unidad mixta de investigación Anfaco-Emenasa trabajan para importar soluciones competitivas a la ingeniería alimentaria basándose en las líneas de investigación de fábrica conectada, robótica avanzada y producción individualizada. En concreto con el proyecto europeo SpecTuna, que se trata de una línea de caracterización de la humedad y la grasa del atún [58].

Figura 29: Proyecto europeo SpecTuna [58]

- Proyecto Blue Fish: la empresa gallega TACORE, a través de un proyecto de I+D, mediante visión artificial ha desarrollado un equipo de clasificación de tamaños y especies de peces para su conserva. El sistema alcanza una producción entorno a las 1000 unidades por minuto.

- Envasado de pimientos tricolor: uno de los primordiales mayoristas de verduras y frutas de Holanda, instaló una nueva línea de envasado de pimientos tricolor asistida por robots que aumentó un 30% la productividad [59].

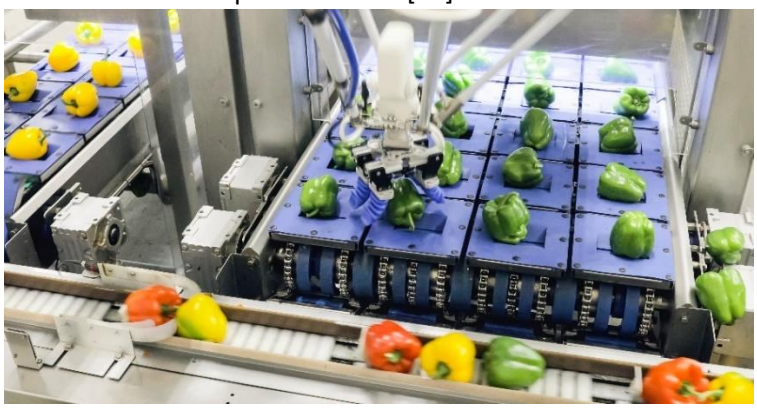


Figura 30: Envasado de pimientos tricolor [60]

7.2. Aspectos fundamentales de la robótica

La definición de robot industrial según la norma internacional ISO 8373:2012 es [61]:

"Un manipulador funcional, reprogramable y controlado automáticamente, programable en tres o más ejes que puede estar fijo en un área o móvil para su uso en aplicaciones de automatización industrial."

Los robots se caracterizan por su diversidad en términos de geometría, velocidad y espacio de trabajo. Se presenta a continuación, su clasificación según geometría y ordenados cronológicamente por su popularidad:

- Articulados: son ampliamente utilizados y altamente versátiles. Son capaces de soportar cargas de hasta una tonelada. Además, cuentan con un amplio espacio de trabajo.

Figura 31: Robot articulado Sepro Yaskawa 6X-140 [62]

- Scara: entre sus características destaca que son muy rápidos, pero suelen tener un espacio de trabajo limitado en el eje Z, pero amplio en el eje X, Y. Son los robots más utilizados para tareas de Pick and Place (recoger y colocar).

Figura 32: Robot Epson Scara LS10-B703S [63]

- Delta: son los robots más rápidos, pero con un alcance muy limitado en todos los ejes. Suelen encontrarse colgados del techo, y suelen realizar operaciones de Pick and Place

Figura 33: Robot Delta C-AX050 [64]

Los robots colaborativos (CoBots) son robots industriales diseñados para interaccionar directamente con un humano dentro de un espacio de trabajo cooperativo, sin que sea necesario la existencia de un espacio de seguridad aislado [65].

7.3. Características del brazo robot colaborativo utilizado

El cobot utilizado ha sido el xArm 6 de UFactory facilitado por la empresa Premo Robotics, consta de una base y seis articulaciones rotativas, y cada articulación representa un grado de libertad, su velocidad puede alcanzar hasta los 180°/s. El peso del brazo robótico es de 12.2kg.

La hoja de características puede verse en Anexo - Fichas Técnicas.

La siguiente figura, muestra el brazo robot utilizado:

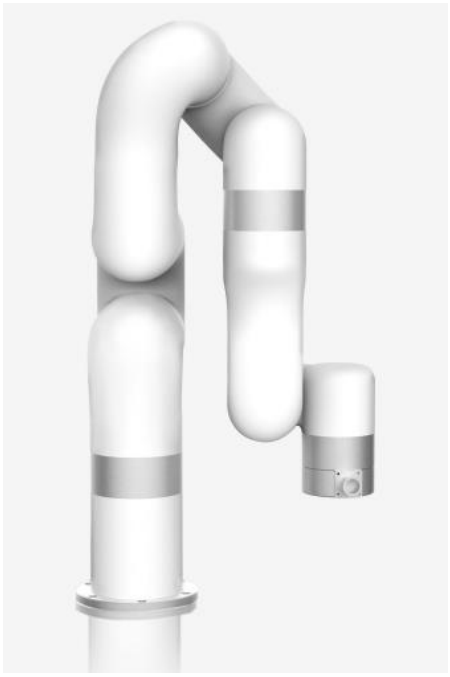


Figura 34: Robot utilizado xArm 6 de UFactory [66]

Desde la parte inferior hasta la superior, en orden, se encuentran la articulación 1, articulación 2, articulación 3 … hasta la última articulación, que es donde se va a conectar un efector final (por ejemplo, una pinza, una pinza de vacío …). El rango de cada articulación es [67]:

- Las articulaciones 1,4,6 es de ±360°
- La articulación 2 es de -118°~120°
- La articulación 3 es de -225°~11°
- La articulación 5 es de -97°~180°

Las figuras que se presentan a continuación muestran los espacios de trabajo:

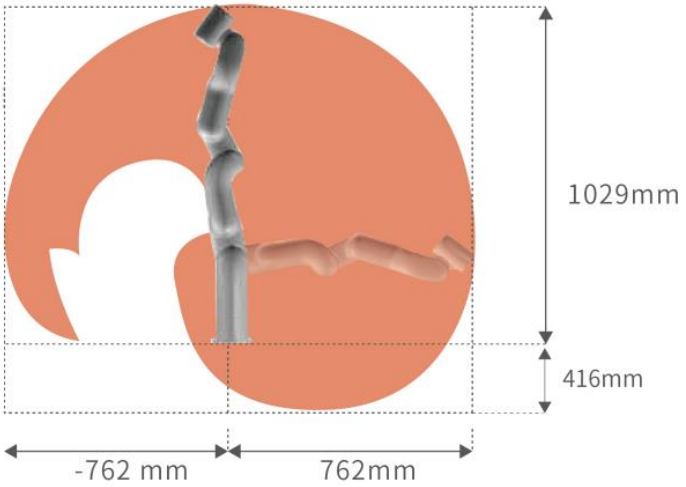


Figura 35: Perfil del rango de trabajo del brazo robótico [67]

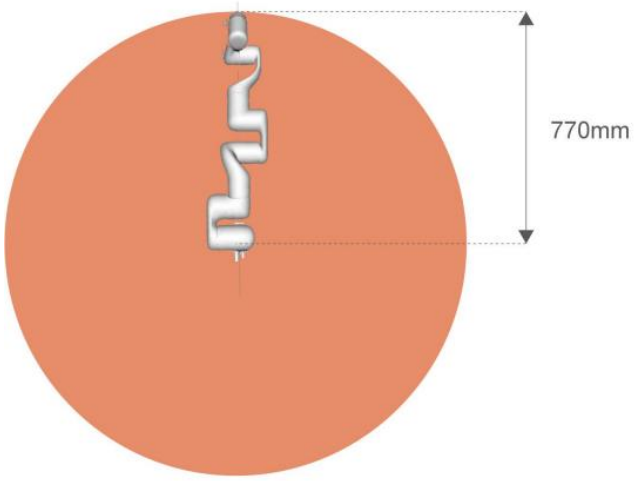


Figura 36: Planta del rango de trabajo del brazo robótico [67]

8. Análisis de los distintos tipos de gripper, alternativa de diseño y diseño final

En este apartado, se llevará a cabo un estudio exhaustivo sobre los diferentes tipos de grippers disponibles en el mercado actual. Dado que ya se ha introducido el brazo robot con el que se trabajará, se elegirá un gripper compatible con él y que se adapte a las necesidades del producto.

Se examinará detalladamente el proceso de diseño, considerando los requisitos específicos para asegurar un agarre eficiente, seguro y que garantice que no se dañan los calabacines.

8.1. Distintos tipos de gripper

Se presentan a continuación, los distintos tipos de gripper [68]:

- Gripper con pinzas de dos dedos: Este tipo de pinza es la más utilizada para la manipulación de piezas. Las pinzas se adaptan fácilmente a la geometría de los objetos y generalmente son tipo paralelo o planas, aunque también existen pinzas angulares.

Figura 37: Pinzas de dos dedos [69]

- Gripper de tres dedos: Este sistema es muy parecido al de las pinzas de dos dedos, pero se diferencia en que las pinzas de tres dedos ofrecen una mayor sujección al agarrar piezas. Este tipo de garra se utiliza cuando se requiere una mayor precisión en la manipulación de productos.

Figura 38: Gripper de tres dedos [70]

- Gripper con pinzas de dedos flexibles o blandos: Estas soluciones de sujeción son novedosas, ya que no existían en el mercado hasta hace pocos años. Son muy adecuadas para productos que necesitan un alto grado de sensibilidad de manipulación.

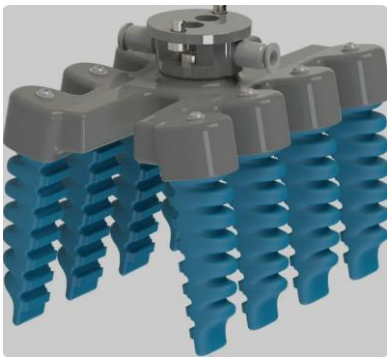


Figura 39: Gripper con pinzas de dedos flexibles o blandos

- Gripper con forma de bola flexible: Los gripper fabricados con materiales blandos pueden tener forma de bola para aumentar su agarre. Por lo general, están hechos de materiales de látex y contienen gránulos en su interior. El sistema de agarre es sencillo, para ello se crea una succión de vacío en el interior del gripper, lo que permite succionar el objeto deseado.

Figura 40: Gripper con forma de bola flexible [72]

- Gripper con pinzas de vacío: Este tipo de gripper es un sistema compuesto por ventosas que permiten mover piezas haciendo uso de vacío. Esta especialmente indicado para agarrar objetos metálicos, cartón, plástico y vidrio. Dependiendo de las necesidades, también se pueden integrar pinzas de vacío eléctricas.

Figura 41: Gripper con pinza de vacío [73]

- Gripper con pinzas para paletización. Por ejemplo, las pinzas usadas para el paletizado de sacos se emplean en células de paletizado y pueden incorporar múltiples sistemas de accionamiento, ya sean eléctricos, hidráulicos o neumáticos. Se caracterizan por ser personalizadas para cada proceso, al igual que las garras utilizadas en robots de soldadura.

Figura 42: Gripper con pinzas para paletización [74]

8.2. Alternativa de diseño

El gripper que lleva instalado el robot colaborativo es del tipo pinzas con dos dedos, se presenta en la figura adjunta:

Figura 43: Gripper instalado [67]

La hoja de características puede verse en Anexo - Fichas Técnicas.

A raíz de ese gripper, se ha diseñado un acople a ese diseño para asegurar el correcto y perfecto transporte de los calabacines.

En la siguiente imagen, se muestra las medidas de la pinza, en mm:

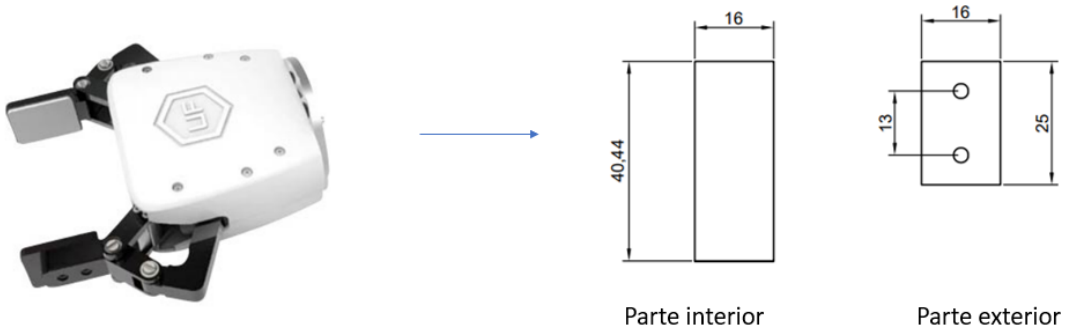


Figura 44: Acotaciones de la pinza, en mm

El espesor sin espuma es de 5.10mm.

La alternativa de diseño fue un gripper de dos dedos. Se utilizó el software SolidWorks para llevar a cabo el proceso. Se crearon piezas individuales que incluyeron el perfil de la pinza, una parte de unión, así como el interior y el exterior de las pinzas. Estas piezas fueron ensambladas para formar una pinza completa. En la figura adjunta, se muestra el diseño preliminar con la distribución de piezas:

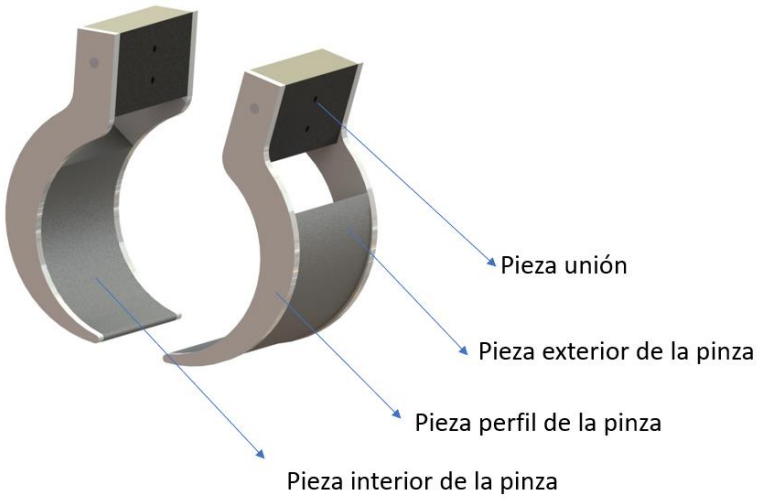


Figura 45: Distribución de piezas de la alternativa de diseño

Dado que la pinza consta de dos piezas individuales idénticas, se aplicó una duplicación por simetría para obtener la pieza final. Esta pieza final también fue ensamblada, existiendo relleno entre la pieza exterior e interior.

La siguiente figura, muestra el renderizado de la pinza:

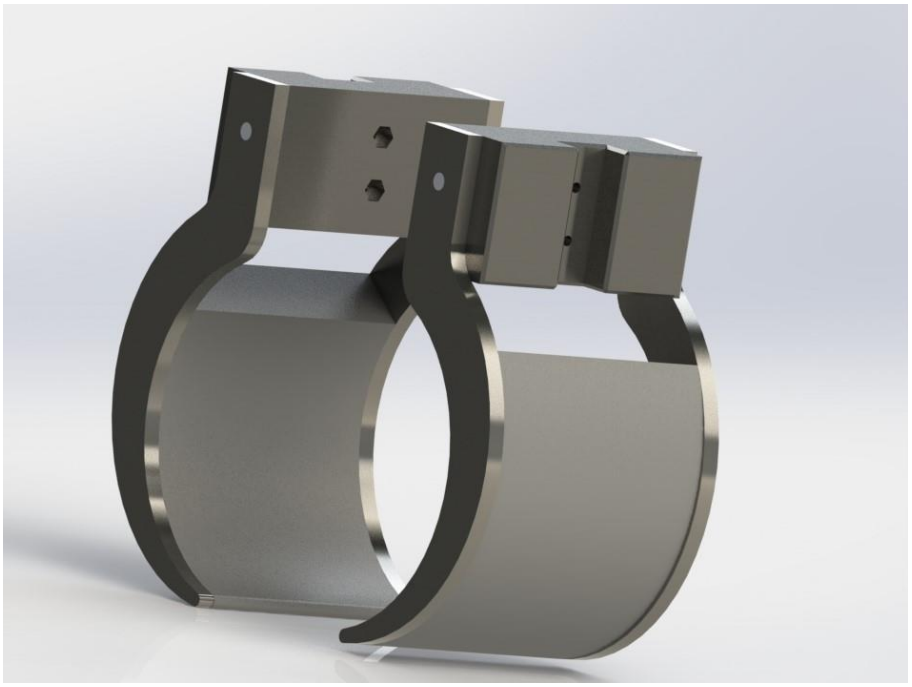


Figura 46: Render de la alternativa de diseño

Antes de proceder a la impresión, se realizó una nueva versión de la pinza con las mismas características, pero con hueco entre la pieza exterior e interior para ahorrar material.

La figura adjunta representa la pieza de unión fabricada mediante impresión 3D. El material usado fue Draft Resin, que se caracteriza por la rápida impresión frente a otras resinas haciéndola así ideal para prototipos iniciales.

Se evidencia que las dimensiones del gripper no eran apropiadas, lo que impediría un funcionamiento adecuado al no permitir una apertura/cierra correctos. Estas fueron unas de las razones por las cuales se descartó este diseño, además de presentar un gran inconveniente en la fijación de las piezas una vez creadas.

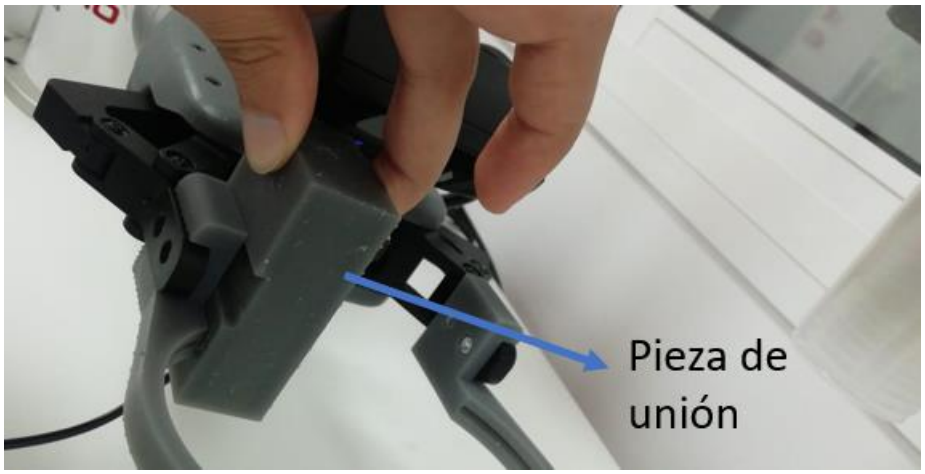


Figura 47: Diseño 3D de la alternativa de diseño

8.3. Desarrollo del diseño final

Después de estudiar detenidamente la alternativa de diseño, se decidió avanzar con una solución modificada pero basada en el mismo concepto. Para esta solución, se usó el software Fusión 360 de Autodesk. Como resultado, se desarrolló un gripper de una sola pieza hueca, manteniendo las medidas facilitadas en la Figura 44: Acotaciones de la pinza, en mm

Se presenta el renderizado de la pieza:

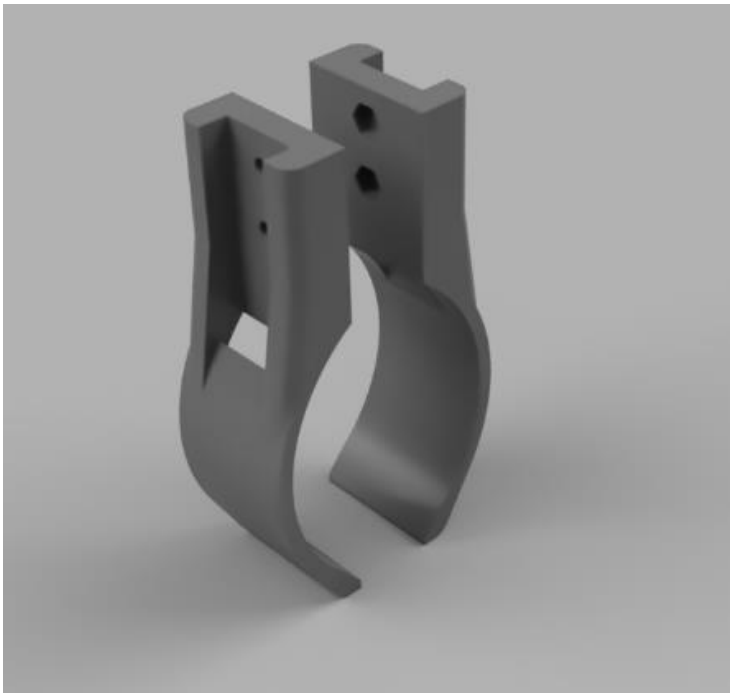


Figura 48: Renderizado de la solución adoptada (perspectiva uno)

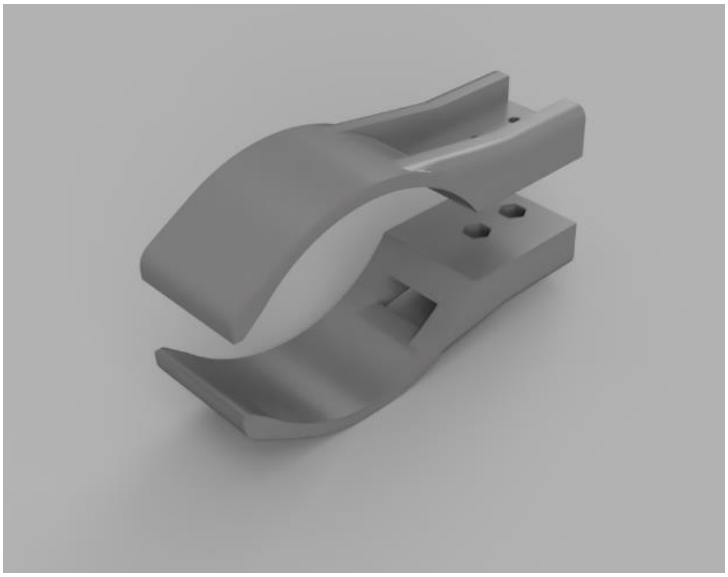


Figura 49: Renderizado de la pieza adoptada (perspectiva dos)

A continuación, se procedió a imprimirla en 3D, también con resina Draft, y se incorporó al brazo robot.

Las siguientes imágenes muestran dicha incorporación, y el agarre que proporciona a los calabacines:

Figura 50: Diseño final de gripper (perspectiva uno)

Figura 51: Diseño final del gripper (perspectiva dos)

Las marcas presentes en la pieza se deben al proceso de lijado realizado para evitar forzar su incorporación.

Figura 52: Diseño del gripper con calabacín perspectiva uno

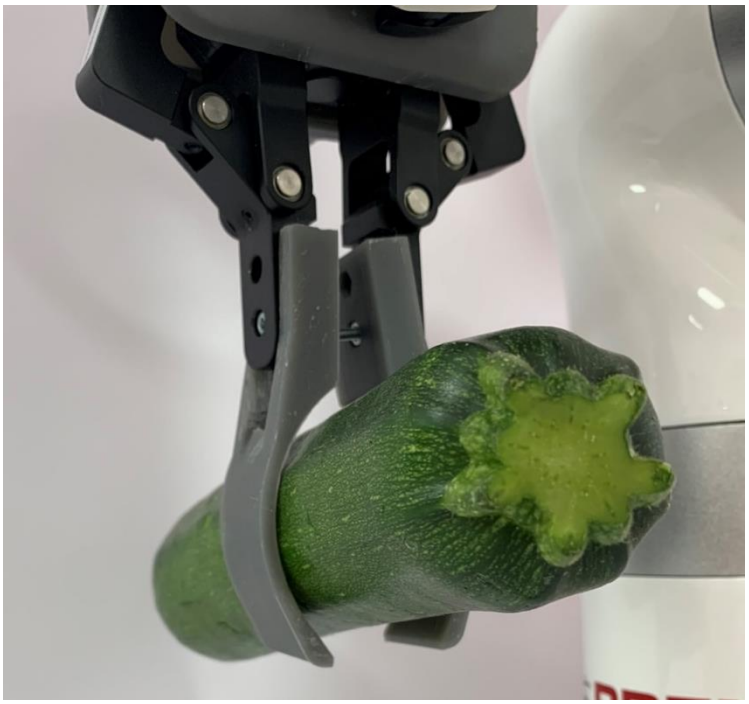


Figura 53: Diseño del gripper incorporado con calabacín (perspectiva dos)

9. Desarrollo del software de manipulación y empaquetado.

En este apartado, se aborda el software de manipulación que complementa el diseño del gripper. Una vez incorporada la pinza diseñada, se realizará el software que colocará los calabacines en tres ubicaciones distintas, dependiendo del calibre que le haya indicado la CNN.

Figura 54: Página principal de U Factory Studio [67]

U Factory Studio es el software utilizado para programar el robot, en la imagen superior se presenta su página principal. La programación del robot se puede llevar a cabo de dos formas distintas, para eso se accede a "Live Control" en la página principal del software.

- La primera opción se puede ver en la figura adjunta, en la parte superior derecha se puede manipular las seis articulaciones.

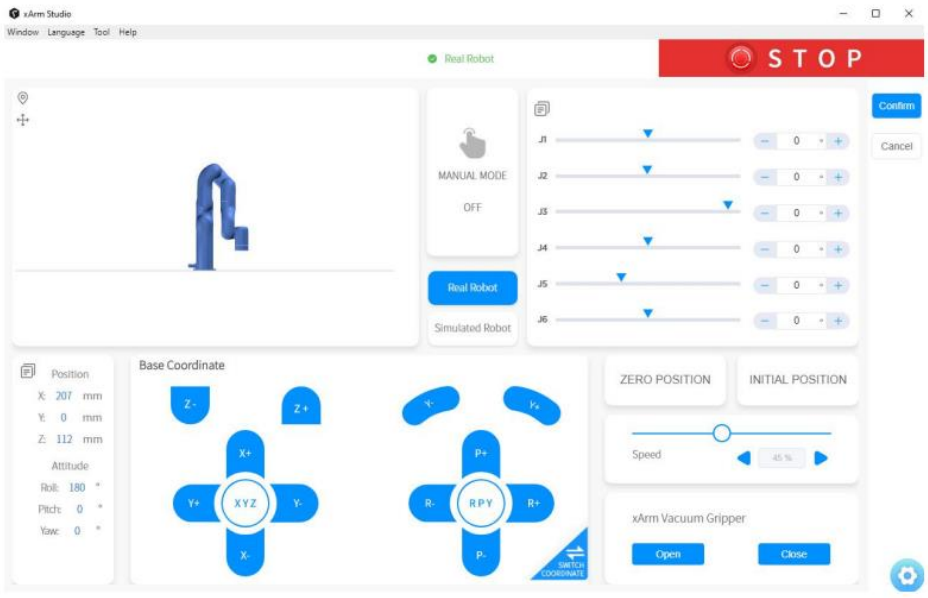


Figura 55: Como se programa el brazo robot de U Factory Studio [67]

- La segunda opción implica el modo manual, para eso, se selecciona “Mode Manual”, que se encuentra en el centro de la imagen proporcionada, y se mueve el robot a la posición deseada de manera manual. Una vez alcanzada la posición deseada, se guarda dicho movimiento.

En este proyecto, en concreto, se ha usado el modo manual para colocar el robot en las tres ubicaciones distintas, y también se ha configurado de modo manual la posición inicial.

La posición inicial es aquella desde la cual se capturará la imagen del calabacín a la misma altura que se utilizó para crear la base de datos. Para realizar esa imagen se ha usado una webcam, cuya ficha de características está en los anexos. En la siguiente figura, se muestra la webcam unida al gripper en la posición inicial.

Figura 56: Posición inicial

Una vez que se han configurado todas las posiciones, se accede al menú de bloques para realizar la configuración del gripper y establecer las condiciones de movimiento que debe seguir en función del tamaño. En este menú, se definen las instrucciones específicas para el gripper, como la apertura o cierre de la pinza, la velocidad del movimiento y cualquier otra acción necesaria para manipular los objetos. Estas configuraciones permiten al sistema robotizado adaptarse a diferentes tamaños de objetos y realizar movimientos específicos según las necesidades del proceso.

A continuación, se accede a “Blocky” en la página principal. Se muestra en la figura adjunta el diagrama de bloques, el cual, aunque no corresponde al programa diseñado, proporciona aclaraciones sobre la configuración del gripper con el nombre de “set xarm gripper” en azul oscuro y las condiciones de actuación que se encuentran nombrados como “Remark” en rojo, que se han usado para diferenciar los tres movimientos distintos del robot dependiendo del calibre del calabacín.

Puesto que se ha configurado el robot de forma manual, en el diagrama de bloques se han modificado los valores para que sean valores exactos.

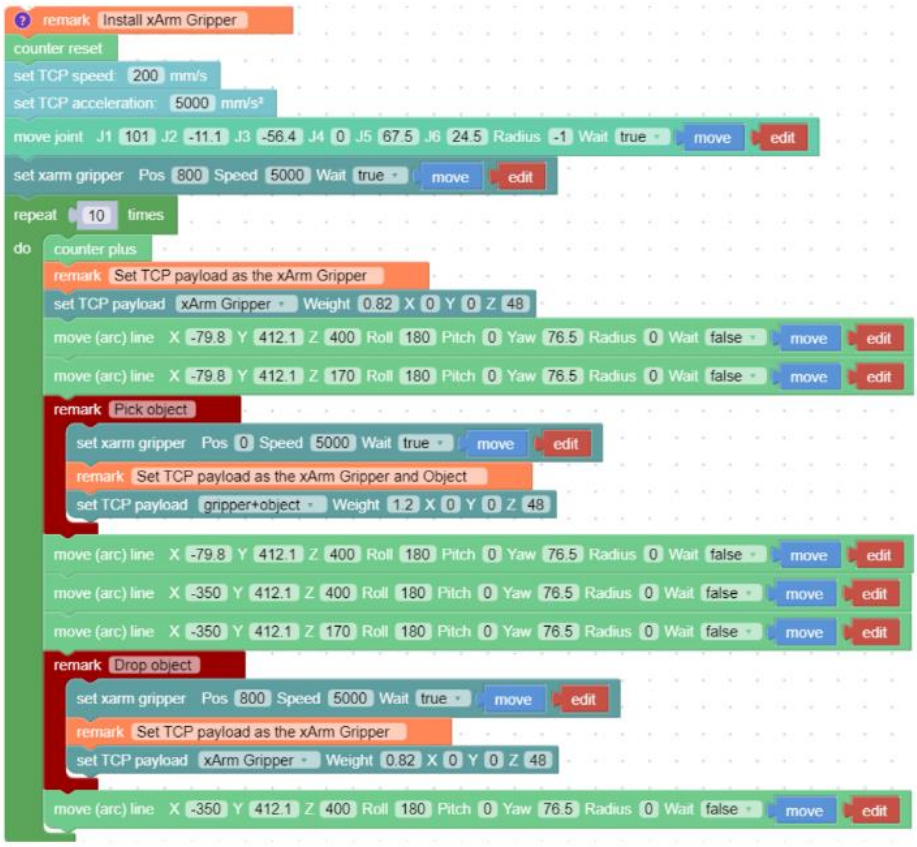


Figura 57: Ejemplo de diagrama de bloques de U Factory Studio [67]

Ahora que ya están configurados todos los movimientos, en la página principal se accede a "Python IDE", donde se encuentra el código Python con todas las acciones generadas y bibliotecas.

10. Integración y pruebas funcionales.

En este apartado, se integra el código generado con los movimientos del robot junto a la CNN creada en apartados anteriores. El código completo se encuentra en los anexos para su referencia.

En el proceso de integración se llevó a cabo los siguientes pasos:

- Se importan las librerías y se inicia el robot.
- Se crea una carpeta utilizando la fecha y hora actual para guardar la foto que será analizada por la CNN. Se determina también la ruta de la carpeta para después ubicarla.
- Se realiza el preprocesado de datos, que es necesario para adoptar la información al máximo para facilitar el entramiento y aprendizaje de la red neuronal.
- 'model = tf.keras.models.load_model('my_model.h5') carga el modelo previamente guardado en el entrenamiento de la CNN. El archivo my_model.h5 contiene tanto la arquitectura como los pesos y configuraciones del optimizador.
- Se inicial el bucle while que se ejecuta un número determinado de veces, dependiendo de las iteraciones deseadas. También está definida la posición inicial que es de donde se toma la foto. Dentro de este bucle, se encuentran tres condiciones if que dependen del tamaño de los calabacines almacenados en la variable` y_val_hat[i] '
  - if 'y_val_hat[i]==0', la CNN ha detectado un calabacín pequeño, por tanto, el brazo robot colaborativo lo colocará en una ubicación solo para pequeños.
  - if 'y_val_hat[i]==1', la CNN ha detectado un calabacín mediano, por tanto, el brazo robot colaborativo lo colocará en una ubicación solo para medianos.
  - if 'y_val_hat[i]==2', la CNN ha detectado un calabacín grande, por tanto, el brazo robot colaborativo lo colocará en una ubicación solo para grandes.
- Tras finalizar el bucle se eliminan las carpetas.

Para realizar las pruebas funcionales, se ha unido la cámara webcam al gripper, como se indicó anteriormente, para tomar la foto desde la posición inicial.

En una cartulina, se realizaron las siguientes anotaciones con el fin de distinguir las tres ubicaciones correspondientes a los calabacines de diferentes tamaños: P para calabacines pequeños, M para calabacines medianos, y G para calabacines grandes.

En el siguiente enlace, se presenta un video de la integración total y las pruebas funcionales:

https://drive.google.com/drive/folders/1HIAwueH_vmU7oexIGWgNnca5jD8fvE7y?usp=drive_link

11. Presupuesto y amortización

En esta sección, se proporciona un desglose presupuestario para el desarrollo del presente proyecto, con el objetivo de posteriormente facilitar una comparación de amortización con el desempeño de una persona realizando la misma tarea.

Para esto, se han considerado las siguientes suposiciones:

- El robot constará como adquirido, pero la impresora 3D usada no.
- Puesto que la impresora no se adquirido, se ha subcontratado a una empresa ajena la impresión del gripper [76].

Ud	Descripción	Precio	Cantidad	Total
litro	Material Draft	163,35 €	0,00283256	0,46 €
ud	Impresión 3D pinzas	13,15 €	2	26,30 €
ud	Gastos de envío piezas	8,47 €	1	8,47 €
ud	Webcam	6 €	1	6,00 €
ud	Brazo robot colaborativo	10.164,21 €	1	10.164,21 €
			Subtotal [€]	10.205,44 €
			IVA (21%) [€]	2.143,14 €
			Total [€]	12.348,59 €

Tabla 4: Presupuesto

Para la amortización del proyecto comparados con los costos relacionados con el empleo de una persona. Se supone lo siguiente:

- Basándose en la velocidad programada, en este proyecto, el robot envasa 5 calabacines por minuto.
- Una persona envasa 10 calabacines por minuto.
- El horario de una persona es de 8:00 a 14:00 y de 16:00 a 18:00, y de 18:00 a 20:00 en valor de hora extra.
  Se establece una jornada de 10 horas, dado que se va a suponer que se encuentra en alta temporada.
  No se contemplan gastos de contratación.
  Se presenta, en la siguiente tabla el desglose de precio según el Convenio colectivo del manipulado de frutas, hortalizas y patata en Granada y provincia [75].

Descripción	Cantidad	Precio	Total horas
Precio hora	8	7,94	63,52
Valor hora extra	2	10,6	21,2
		Total día [€]	84,72 €

Tabla 5: Precio de una persona trabajando al día

- No se contempla el consumo de energía del robot.
- Se considera la campaña del calabacín, al periodo de tiempo específico durante el cual se llevan a cabo actividades agrícolas relacionadas con el cultivo de este. En este proyecto, se ha considerado de tres meses.
- Se considera la precisión y calidad del envasado realizado por el robot igual que el de una persona.

- Se supone que el robot tiene una vida útil de 5 años y que se utiliza durante toda la temporada sin interrupciones, es decir, durante tres meses, veinticuatro horas al día.
- No se considera el mantenimiento, ni el reemplazo de piezas del robot.

Se calcula el tiempo de amortización:

Días laborales de campaña: 5 días a la semana x 4 semanas x 3 meses = 60 días

Ahorro anual = Costo de una persona por día x día laborales de campaña.

Ahorro anual = 84,72 €/día x 60 días

Ahorro anual = 5.083,2€

Tiempo de amortización = Costo del robot / Ahorro anual

Tiempo de amortización = 12,348.59€ / 5.083,2€

Tiempo de amortización ≈ 2,429 campañas

Por lo tanto, el tiempo estimado para amortizar el robot es de 2 campañas y media.

A modo de conclusión, aunque el robot puede ser menos productivo que una persona en términos de velocidad de envasado presenta ventajas significativas en términos de disponibilidad y flexibilidad. El robot puede trabajar durante todas las horas necesarias sin fatiga ni descanso, lo que permite un funcionamiento continuo y una mayor capacidad de producción. Además, se puede ajustar la velocidad del robot para aumentar su rendimiento si es necesario.

Esta conclusión resalta la viabilidad económica de invertir en el robot a pesar de su menor productividad en comparación con una persona. Después de dos campañas y medias, los beneficios económicos generados por el ahorro en costos superarían el costo inicial del robot, lo que resultaría en una inversión rentable a largo plazo.

12. Conclusión y líneas futuras

Durante el desarrollo de este proyecto, documentado en este informe, se han abordado diversas etapas relacionadas con la implementación de algoritmos basados en aprendizaje profundo para la clasificación del calibre de los calabacines.

En primer lugar, se realizó una exhaustiva explicación teórica de los conceptos fundamentales que engloban las disciplinas de la inteligencia artificial, aprendizaje profundo, aprendizaje automático y redes neuronales. Posteriormente, se presentó el entorno de trabajo utilizado y las herramientas necesarias, así como la implementación de la red neuronal convolucional para la clasificación. Aunque solo se ha mostrado el código que ha demostrado ser efectivo, se llevaron a cabo varios experimentos donde el test accurracy era inferior al 60%.

Tras obtener un buen modelo de CNN, se presentó al robot colaborativo al que se le diseño un gripper que garantizase el buen transporte de los calabacines. Se programo éste para que depositase los calabacines en tres ubicaciones distintas, dependiendo del tamaño.

Se realizaron pruebas funcionales, corroborando el correcto funcionamiento de todo el proyecto.

12.1 Líneas futuras

Como línea futura, se propone ampliar la base de datos con una mayor variedad de calabacines en términos de tamaño y forma. Además, se sugiere incluir calabacines de segunda calidad, que presenten cortes, fricciones o daños leves en el pedúnculo, entre otras características. Para adaptarse a estas características, se recomienda implementar una interfaz de usuario que permita parametrizar estos aspectos, ya que son aspectos variables y que dependen de la demanda del calabacín.

También se plantea la posibilidad de utilizar cámaras hiperespectrales para distinguir las diferentes etapas de maduración de los calabacines y realzar una clasificación basada en este parámetro.

Además, se sugiere explorar la aplicación de técnicas como YOLO. YOLO es un sistema de detección de objetos capaz de detectarlos en tiempo real , sería una opción para implementar un sistema automatizado en una cinta transportadora en movimiento.

Referencias

[1] ¿Qué es la Visión artificial? (s/f). Intel. Recuperado el 27 de mayo de 2023, de https://www.intel.es/content/www/es/es/manufacturing/what-is-machine-vision.html

[2] Hernández, R., Ortigoza, R. S., Vilchis, R. M., & Aurora, M. (2007). La Visión Artificial en la Robótica. Redalyc.org. https://www.redalyc.org/pdf/4026/402640448005.pdf

[3] Monteoliva, G. (2022, octubre 10). Visión artificial en el sector de la alimentación. INFAIMON. https://infaimon.com/alimentacion/vision-artificial-sector-alimentacion/

[4] BOE.es - DOUE-L-1981-80141 Reglamento (CEE) no 1292/81 de la Comisión, de 12 de mayo de 1981, por el que se establecen normas de calidad para los puerros, las berenjenas y los calabacines. (s/f). Boe.es. Recuperado el 27 de mayo de 2023, de https://www.boe.es/buscar/doc.php?id=DOUE-L-1981-80141

[5] Breve historia de la inteligencia artificial: el camino hacia la empresa. (s/f). Cesce España. Recuperado el 27 de mayo de 2023, de https://www.cesce.es/es/w/asesores-de-pymes/breve-historia-la-inteligencia-artificial-camino-hacia-la-empresa

[6] Roa, J. (s/f). inteligencia artificial. Blogspot.com. Recuperado el 27 de mayo de 2023, de http://josegerma.blogspot.com/2012/06/en-la-decada-de-los-60.html

[7] Disruption from AI and Business Analytics. (s/f).

[8] López, R et al. (2021). Inteligencia Artificial y Machine Learning en trastornos del movimiento. MANUAL SEN 2021.

[9] Dalal, K.R. (2020). Analysing the Role of Supervised and Unsupervised Machine Learning in IoT. International Conference on Electronics and Sustainable Communication Systems (ICESC).

[10] Dalal, K.R. (2020). Analysing the Role of Supervised and Unsupervised Machine Learning in IoT. International Conference on Electronics and Sustainable Communication Systems (ICESC).

[11] Tian, Y. and Compere M. (2019). A Case Study on Visual-Inertial Odometry using Supervised, SemiSupervised and Unsupervised Learning Methods. 2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), pp. 203-2034

[12] Oquab, M. et al. (2015). Is object localization for free? - weakly-supervised learning with convolutional neural networks. Proceedings of the IEEE conference on computer vision and pattern recognition.

[11] Ren, H. et al. (2020). Stereo Disparity Estimation via Joint Supervised, Unsupervised, and Weakly Supervised Learning, 2020 IEEE International Conference on Image Processing (ICIP), pp. 2760-2764-

[12] Reese, H. (2017). Understanding the differences between AI, machine learning, and deep learning. URL: https://www. techrepublic. com/article/understandingthedifferencesbetweenaimachine learninganddeeplearning

[13] Lauzon, FQ. (2012) An introduction to deep learning. 2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA). IEEE

[14] Cirugía Robótica. (s/f). Cirugiaroboticaha.com. Recuperado el 27 de mayo de 2023, de https://cirugiaroboticaha.com/articulo.php?art=23

[15] Pérez, E. (2020, enero 31). La carretera a ojos del Autopilot de Tesla: esto es todo lo que capta su sistema de conducción autónoma. Xataka.com; Xataka. https://www.xataka.com/vehiculos/carretera-a-ojos-autopilot-tesla-esto-todo-que-capta-su-sistema-conduccion-autonoma

[16] Sandra. (2021, mayo 4). 5 aplicaciones de detección de defectos por visión artificial. ATRIA Innovation. https://www.atriainnovation.com/deteccion-de-defectos-por-vision-artificial/

[17] OPTINVAS: Inteligencia artificial aplicada a procesos de calidad y envasado para generar soluciones de trazabilidad. (s/f). Espuna.es. Recuperado el 27 de mayo de 2023, de https://www.espuna.es/optinvas-inteligencia-artificial-aplicada-a-procesos-de-calidad-y-envasado-para-generar-soluciones-de-trazabilidad

[18] Detección de objetos mediante Fast R-CNN - Cognitive Toolkit - CNTK. (s/f). Microsoft.com. Recuperado el 27 de mayo de 2023, de https://learn.microsoft.com/es-es/cognitive-toolkit/object-detection-using-faster-r-cnn

[19] Guzman, E. R. S. (s/f). PROGRAMA DE MAESTRIA Y DOCTORADO EN INGENIERIA INGENIERIA ELECTRICA - PROCESAMIENTO DIGITAL DE SENALES. Unam.mx. Recuperado el 27 de mayo de 2023, de https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/reconocimiento_de_patrones/tutoriales/YOLO-Introducci%C3%B3n-e-implementaci%C3%B3n-.pdf

[20] 330ohms. (2020, noviembre 17). ¿Cómo detectar objetos con YOLO? 330ohms. https://blog.330ohms.com/2020/11/17/deteccion-de-objetos-con-yolo/

[21] Ghazimirsaeed, S.M et al. (2020). Accelerating GPU-based Machine Learning in Python using MPI Library: A Case Study with MVAPICH2-GDR. IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC) and Workshop on Artificial Intelligence and Machine Learning for Scientific Applications (AI4S).

[22] Ketkar, N. (2017) Introduction to Deep Learning. In: Deep Learning with Python. Apress, Berkeley, CA. [16] Illingworth, W.T. (1989). Beginner's guide to neural networks. IEEE Aerospace and Electronic Systems Magazine, vol. 4, no. 9, pp. 44-49.

[23] Aldabas-Rubira, E. (2002). Introducción al reconocimiento de patrones mediante redes neuronales. IX Jornades de Conferències d'Enginyeria Electrònica.

[24] Tandeitnik, P. and Guterman, H (1996). Modelling of biological neurons by artificial neural networks, Proceedings of 19th Convention of Electrical and Electronics Engineers in Israel, pp. 239-242.

[25] Steck,J.E. and Dalton J.S. (1992). Modeling the neurodynamics of a biological neuron using a feedforward

[26] Steck,J.E. and Dalton J.S. (1992). Modeling the neurodynamics of a biological neuron using a feedforward artificial neural network. IJCNN International Joint Conference on Neural Networks

[27] Tablada, C.J y Torres G.A. (2009). Redes neuronales artificiales. Revista de educación matemática 24.3. [21] Santillán, D. et al. (2014). Predicción de lecturas de aforos de filtraciones de presas bóveda mediante redes neuronales artificiales. Tecnología y ciencias del agua.

[28] Gong, D. et al. (2020). Learning Deep Gradient Descent Optimization for Image Deconvolution. IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 12, pp. 5468-5482.

[29] Moreno, A. (1994). Aprendizaje automático. Universidad Politécnica de Cataluña, Edicions UPC. [25] Zeiler, M.D. (2012). An adaptive learning rate method. Adadelta.

[30] ¿Qué es la entropía cruzada en Deep Learning? (2022, agosto 16). KeepCoding Bootcamps. https://keepcoding.io/blog/entropia-cruzada-deep-learning/

[31] Han, X. et al. (2016). An FPGA-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks. IEEE 34th International Conference on Computer Design (ICCD).

[32] Liu, R. et al. (2019). Learning Converged Propagations With Deep Prior Ensemble for Image Enhancement. IEEE Transactions on Image Processing, vol. 28, no. 3, pp. 1528-1543.

[33] Fooladi, N. et al. (2014). Permeability Prediction of an Iranian Reservoir Using Hybrid Neural Genetic Algorithm.

[34] Torres, L. G. (1994). Redes neuronales y aproximación de funciones. Bol. Matemáticas, 1, 35-58.

[35] Funciones de activación y de costo (parte 1) · Aprendizaje Profundo. (s/f). Github.io. Recuperado el 27 de mayo de 2023, de https://atcold.github.io/pytorch-Deep-Learning/es/week11/11-1/

[36] Radečić, D. (2020, junio 18). Softmax activation function explained. Towards Data Science. https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60

[37] Setiono, R. y Liu, H. (1997). Neural-network feature selector. IEEE Transactions on Neural Networks, vol. 8, no. 3, pp. 654-662.

[38] Bullinaria, J.A. (2015). Bias and variance, under-fitting and over-fitting. Neural Computation: Lecture 9.

[39] Koehrsen, W. (2018). Overfitting vs. underfitting: A complete example. Towards Data Science.

[40] Dileep, D. y Bora P.K. (2020). Dense Layer Dropout Based CNN Architecture for Automatic Modulation Classification. 2020 National Conference on Communications (NCC).

[41] Srivastava, N. (2013). Improving neural networks with dropout. University of Toronto 182.566: 7.

[42] Sanjar, K. et al. (2020). Weight Dropout for Preventing Neural Networks from Overfitting. 8th International Conference on Orange Technology (ICOT).

[43] ¿Qué es Epoch en Machine Learning? (2021, diciembre 15). Ciberseguridad. https://ciberseguridad.com/guias/nuevas-tecnologias/machine-learning/epoch/

[44] Sandoval Serrano, L.J. (2018). Algoritmos de aprendizaje automático para análisis y predicción datos. Revista Tecnológica; no. 11.

[45] Aloysius, N. y Geetha, M. (2017). A review on deep convolutional neural networks. 2017 International Conference on Communication and Signal Processing (ICCSP).

[46] Masci, J. et al. (2012). Steel defect classification with Max-Pooling Convolutional Neural Networks. The 2012 International Joint Conference on Neural Networks (IJCNN).

[47] Wu, H. y Xiaodong G. (2015). Max-pooling dropout for regularization of convolutional neural networks. International Conference on Neural Information Processing. Springer.

[48] Abirami, S., & Chitra, P. (2020). Energy-efficient edge based real-time healthcare support system. En Advances in Computers (pp. 339–368). Elsevier.

[49] Rolon-Mérette, D. et al. (2016). Introduction to Anaconda and Python: Installation and setup.

[50] Lee, W.M. (2019). Python machine learning. John Wiley & Sons.

[51] Singhla, R et al. (2021). Image Classification Using Tensor Flow. 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS).

[52] Shorten, C. and Taghi M.K. (2019). A survey on image data augmentation for deep learning. Journal of Big Data 6.1

[53] Perez, L. y Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621.

[54] Cómo mejorar los modelos de aprendizaje automático con técnicas de aumento de datos (Data Augmentation). (2020, enero 27). Franco Piergallini Guida. https://franspg.dev/2020/01/27/generacion-de-datos-artificiales-data-augmentation/

[55] Estado del arte de automatización y robtica. (s/f). https://www.atiga.es/web/wp-content/uploads/2017/03/Estado-del-Arte-Automatizaci%C3%B3n-y-rob%C3%B3tica.pdf

[56] Número de unidades de robots industriales vendidas en el mundo 2004-2019. (s/f). Statista. Recuperado el 27 de mayo de 2023, de https://es.statista.com/estadisticas/635304/numero-de-unidades-de-robots-industriales-vendidas-a-nivel-mundial/

[57] Dann, L. (2018, agosto 17). Video: Winter wheat harvest success for Hands Free Hectare. Farmers Weekly. https://www.fwi.co.uk/arable/harvest/video-winter-wheat-harvest-success-for-hands-free-hectare

[58] El proyecto SpecTUNA avanza hacia la automatización de las etapas preparatorias de la materia prima en la industria conservera. (2021, marzo 9). Blog CYTMA. https://anfaco.es/blog_ct/index.php/2021/03/09/el-proyecto-spectuna-avanza-hacia-la-automatizacion-de-las-etapas-preparatorias-de-la-materia-prima-en-la-industria-conservera/

[59] Colás, F. (2023). Las 5 principales predicciones de automatización para 2023 de OMRON Europe. Omron.es. https://industrial.omron.es/es/solutions/blog/top-5-automation-predictions-for-2023

[60] Robots en paralelo y un sistema de visión aceleran la línea de envasado de pimientos. (s/f). Interempresas. Recuperado el 27 de mayo de 2023, de

https://www.interempresas.net/FrontPage/Articles/228416-Robots-en-paralelo-y-un-sistema-de-vision-aceleran-la-linea-de-envasado-de-pimientos.html

[61] ISO 8373:2012. (2021). ISO. https://www.iso.org/standard/55890.html

[62] Innova y propone robots de, S. (s/f). Robots de 6 ejes numéricos. Sepro-group.com. Recuperado el 27 de mayo de 2023, de https://www.sepro-group.com/uploads/6X%20YASKAWA%20ES%20%281%29.pdf

[63] Epson SCARA LS10-B602S (estándar). (s/f). Epson.es. Recuperado el 27 de mayo de 2023, de https://www.epson.es/es_ES/productos/robots/serie-scara-ls/epson-scara-ls10-b602s-%28est%C3%A1ndar%29/p/28176

[64] DR Robots Delta. (s/f). Weiss-world.com. Recuperado el 27 de mayo de 2023, de https://www.weiss-world.com/es-es/products/robots-10327/robots-delta-211

[65] Cobots, ¿Qué son y para que sirven? (s/f). Cfzcobots.com. Recuperado el 27 de mayo de 2023, de https://cfzcobots.com/cobots-que-son-y-para-que-sirven/

[66] UFACTORY xArm 6 Robotic Arm. (s/f). Digital Manufacturing Store Top 3D Shop. Recuperado el 27 de mayo de 2023, de https://top3dshop.com/product/ufactory-xarm-6-robotic-arm

[67] (S/f). Ufactory.cc. Recuperado el 27 de mayo de 2023, de http://download.ufactory.cc/xarm/en/xArm%20User%20Manual.pdf?v=1578910898247

[68] RdR. (2020, agosto 7). Catálogo con diferentes tipos de Gripper y Pinzas robóticas. REVISTA DE ROBOTS. https://revistaderobots.com/sistemas-de-agarre/tipos-de-gripper-y-pinzas-roboticas/

[69] OnRobot • Pinza de dedos RG2. (s/f). MEMIDOS. Recuperado el 27 de mayo de 2023, de https://www.memidos.com/es/producto/onrobot-rg2-finger-gripper/

[70] 3FG15: Pinza de tres dedos que optimiza los procesos de mantenimiento de maquinaria. (s/f). OnRobot. Recuperado el 27 de mayo de 2023, de https://onrobot.com/es/productos/3fg15-pinza-de-tres-dedos

[71] SoftGripping. (2021, septiembre 30). SoftGripping. https://soft-gripping.com/es/

[72] Un gripper universal para robots manipuladores basado en un globo de látex y granos de café. (s/f). Blogspot.com. Recuperado el 27 de mayo de 2023, de http://solorobotica.blogspot.com/2012/05/un-gripper-universal-para-robots.html

[73] VG10 Vacuum Gripper. (s/f). ROS Components. Recuperado el 27 de mayo de 2023, de https://www.roscomponents.com/es/pinzas/256-vg10-vacuum-gripper-on-robot.html

[74] Garras mecánicas para paletizado de sacos y bolsas. (s/f). Piab.com. Recuperado el 27 de mayo de 2023, de https://www.piab.com/es-es/news/garras-mecanicas-para-paletizado-de-sacos-y-bolsas/

[75] Manipulado y Envasado de Frutas, Hortalizas y Patata de Granada. (2015, abril 22). Convenios colectivos. https://www.convenioscolectivos.net/manipulado-y-envasado-de-frutas-hortalizas-y-patata-de-granada/

[76] Fabricación de piezas 3D para particulares. (2020, julio 27). Addimen. https://www.addimen.com/impresion-3d-particulares/

Anexo - Fichas Técnicas

Ficha técnica del robot:

xArm		
Cartesian Range	X	±700mm
	Y	±700mm
	Z	-400mm~951.5mm
	Roll/Yaw/Pitch	± 180°
Maximum Joint Speed		180°/s
Reach		700mm
Repeatability		±0.1mm
Max Speed of End-effector		1m/s
*Ambient Temperature Range		0-50 °C*
Power Consumption		Min 8.4 W, Typical 200W, Max 500W
Input Power Supply		24 V DC, 16.5 A
ISO Class Cleanroom		5
Robotic Arm Mounting		Any
Programming		xArm Studio/Python/C++/ROS
Robotic Arm Communication Protocol		Modbus-TCP
End-effector I/O Interface		2 Digital inputs, 2 Digital outputs, 2 Analog inputs
End-effector Communication Protocol		Modbus-RTU
Footprint		Ø 126 mm
Materials		Aluminium, Carbon Fiber
End Tool Flange		DIN ISO 9409-1-A50/63 (M5*6)
Control Box		
	AC Control Box	DC Control Box
Input	100-240VAC 50/60Hz	24VDC
Output	24VDC 20.8A	24VDC 16.5A
Control Box Communication Protocol	Modbus TCP	
Control Box Communication Model	Ethernet	
Control Box I/O Interface	8*CI+8*DI(Digital In)	8*CI(Digital In)
	8*CO+8*DO(Digital Out) 2*AI(Analog In) 2*AO(Analog Out) 1*RS-485 Master 1*RS-485 Slave	8*CO(Digital Out) 2*AI(Analog In) 2*AO(Analog Out)
Weight	3.9kg	1.6kg
Dimension(L*W*H)	285*135*101mm	180*145*68mm

Ficha técnica del Gripper

Gripper	
Nominal Supply Voltage	24V DC
Absolute Maximum Supply Voltage	28V DC
Quiescent Power (Minimum Power	1.5W
Peak Current	1.5A
Working Range	86mm
Maximum Clamping Force	30N
Weight (g)	822g
Communication Mode	RS-485
Communication Protocol	Modbus RTU
Programmable Gripping Parameters	Position, Speed
Feedback	Position

Anexo – Códigos

El primer código que se presenta, Premo_Classic_CNN_Calabacines, es el código perteneciente a la CNN.

El segundo código, Integración CNN + Robot, es el código de integración con el robot y la CNN.

Premo_Classic_CNN_Calabacines

May 27, 2023

#Librerías

Se presentan las librerias y los paquetes que se van a usar para hacer el desarrollo, implementación y análisis del código.

[ ]:
```
import glob
import random as rn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop, Adam
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix


%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

1 Carga de datos

Se procede a conectar/montar google drive con google colab.

[ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

Se especifica la ruta de carpetas (el directorio) donde se encuentra el zip con las imágenes dentro de google drive y descomprimirla.

[ ]:
```
!unzip gdrive/My\ Drive/Colab\ Notebooks/Calabacines.zip > /dev/null
```

Rutas:

[ ]:
```
path = 'Calabacines/'


# Definición de directorios
train_pequeño_dir = path + 'train/Pequeños/'
train_mediano_dir = path + 'train/Medianos/'
train_grande_dir = path + 'train/Grandes/'

test_pequeño_dir = path + 'test/Pequeños/'
test_mediano_dir = path + 'test/Medianos/'
test_grande_dir = path + 'test/Grandes/'


val_pequeño_dir = path + 'val/Pequeños/'
val_mediano_dir = path + 'val/Medianos/'
val_grande_dir = path + 'val/Grandes/'



# Localización de los archivos (extensión .jpeg)
train_pequeño_cases = glob.glob(train_pequeño_dir + '*jpg')
train_mediano_cases = glob.glob(train_mediano_dir + '*jpg')
train_grande_cases = glob.glob(train_grande_dir + '*jpg')

test_pequeño_cases = glob.glob(test_pequeño_dir + '*jpg')
test_mediano_cases = glob.glob(test_mediano_dir + '*jpg')
test_grande_cases = glob.glob(test_grande_dir + '*jpg')

val_pequeño_cases = glob.glob(val_pequeño_dir + '*jpg')
val_mediano_cases = glob.glob(val_mediano_dir + '*jpg')
val_grande_cases = glob.glob(val_grande_dir + '*jpg')


# Creación de listas vacías y asociación de etiquetas
train_list = []
test_list = []
val_list = []

for x in train_pequeño_cases:# con esta parte metemos las fotos en listas
    train_list.append([x, 0])

for x in train_mediano_cases:
    train_list.append([x, 1])
```

```
for x in train_grande_cases:
    train_list.append([x, 2])



for x in test_pequeño_cases:
    test_list.append([x, 0])

for x in test_mediano_cases:
    test_list.append([x, 1])

for x in test_grande_cases:
    test_list.append([x, 2])



for x in val_pequeño_cases:
    val_list.append([x, 0])

for x in val_mediano_cases:
    val_list.append([x, 1])

for x in val_grande_cases:
    val_list.append([x, 2])



# Mezclado
rn.shuffle(train_list)
rn.shuffle(test_list)
rn.shuffle(val_list)


# Se transforman en Dataframes, es un formato estandar para trabajar en formato␣
 ↪tabla
# hay 2 columnas las imagenes en una y  las etiquetas en otra (es decir, 0,1,2␣
 ↪que es lo mismo que pequeño, mediano, grande)
train_df = pd.DataFrame(train_list, columns=['image', 'label'])
test_df = pd.DataFrame(test_list, columns=['image', 'label'])
val_df = pd.DataFrame(val_list, columns=['image', 'label'])
```

[ ]:
```
val_df
```

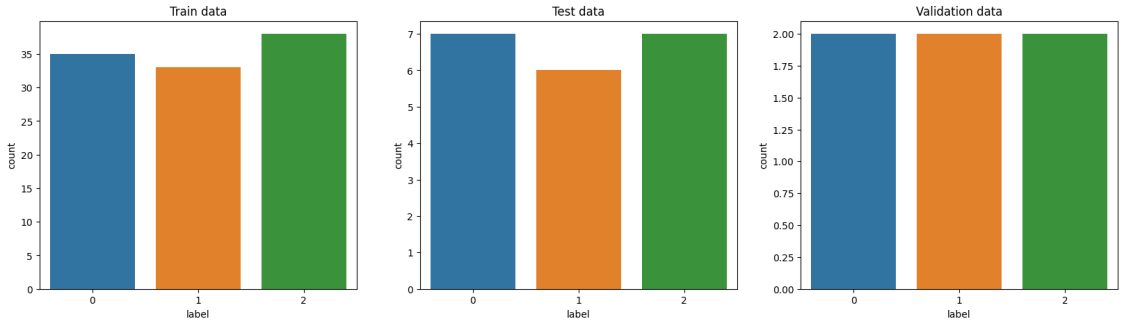
[ ]:

	image	label
0	Calabacines/val/Grandes/Calabacin_124.jpg	2
1	Calabacines/val/Medianos/Calabacin_91.jpg	1

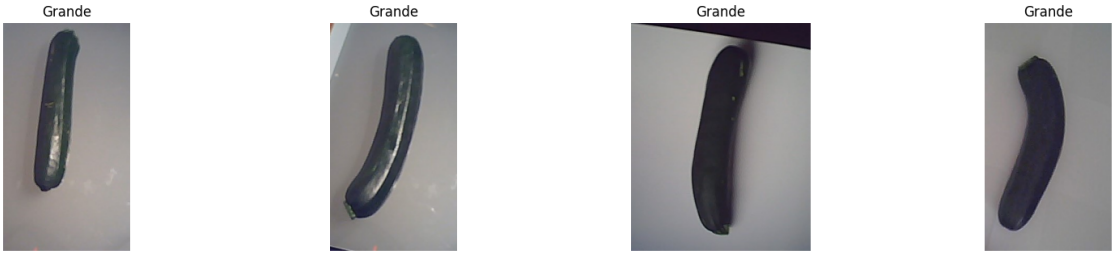
```
2  Calabacines/val/Grandes/Calabacin_125.jpg      2
3   Calabacines/val/Pequeños/Calabacin_7.jpg      0
4   Calabacines/val/Pequeños/Calabacin_3.jpg      0
5  Calabacines/val/Medianos/Calabacin_90.jpg      1
```

2 Visualización de los datos

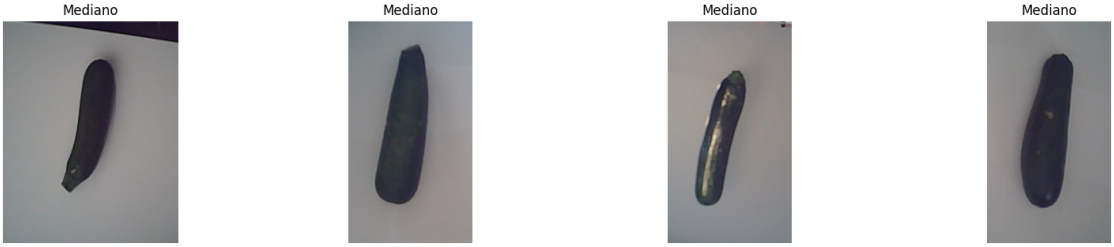
[ ]:
```
plt.figure(figsize=(20,5))#histogramas del conteo de imágenes repartidas en␣
↪cada parte del dataset

plt.subplot(1,3,1)
sns.countplot(x=train_df['label'])
plt.title('Train data')

plt.subplot(1,3,2)
sns.countplot(x=test_df['label'])
plt.title('Test data')

plt.subplot(1,3,3)
sns.countplot(x=val_df['label'])
plt.title('Validation data')

plt.show()
```



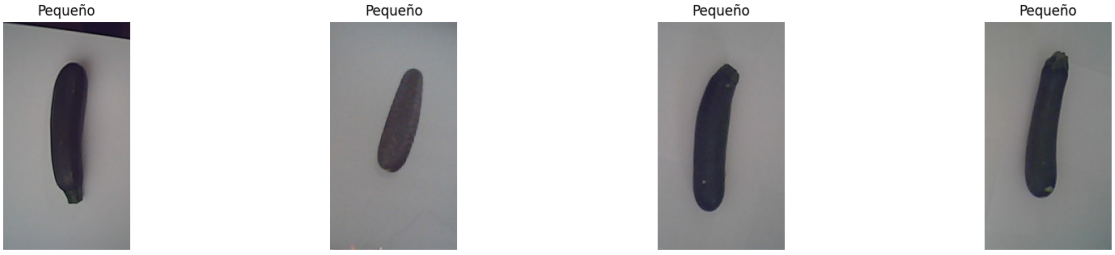
[ ]:
```
plt.figure(figsize=(20,8))#imágenes calabacines grandes
for i,img_path in enumerate(train_df[train_df['label'] == 2][0:4]['image']):
    plt.subplot(2,4,i+1)
    plt.axis('off')
    img = plt.imread(img_path)
    plt.imshow(img, cmap='gray')
    plt.title('Grande')
```



[ ]:
```
plt.figure(figsize=(20,8))#imágenes de medianos

for i,img_path in enumerate(train_df[train_df['label'] == 1][0:4]['image']):
    plt.subplot(2,4,i+1)
    plt.axis('off')
    img = plt.imread(img_path)
    plt.imshow(img, cmap='gray')
    plt.title('Mediano')
```



[ ]:
```
plt.figure(figsize=(20,8))#imágenes de pequeños

for i,img_path in enumerate(train_df[train_df['label'] == 0][0:4]['image']):
    plt.subplot(2,4,4+i+1)
    plt.axis('off')
    img = plt.imread(img_path)
    plt.imshow(img, cmap='gray')
    plt.title('Pequeño')
```



3 Preprocesado de datos

El preprocesado de las imágenes, necesario para adaptar la información al máximo para facilitarle el entrenamiento y aprendizaje a la red neuronal.

[ ]:
```
def process_data(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (196, 196))#se redimensiona la imagen a 196x196␣
 ↪píxeles
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convierte la imagen a escala␣
 ↪de grises, dado que simplifica la detección de patrones
    img = img/255.0 #normalización de la imagen diviendo entre 255 (gama de␣
 ↪colores RGB) para asegurar que los píxeles estan en el rango de 0 a 1
    img = np.reshape(img, (196,196,1)) #reorganiza la forma de la imagen para␣
 ↪que tenga una dimensión adicional de 1  en el eje de los canales,
    #esto se hace porque los modelos CNN generalmente esperan una dimensión␣
 ↪adicional para los canales de color

    return img

def compose_dataset(df):
    data = [] #Creación de dos listas vacías, para almacenar los datos␣
 ↪procesados
    labels = [] #y las etiquetas correspondientes

    for img_path, label in df.values:
        data.append(process_data(img_path)) #Itera sobre cada par de valores
        labels.append(label)

    return np.array(data), np.array(labels) #Convierte las listas data y labels␣
 ↪en matrices numpy
```

En esta celda se presenta información sobre el tamaño y estrucutra de los conjuntos de datos utilizados en el modelo:

[ ]:
```
X_train, y_train = compose_dataset(train_df)
X_test, y_test = compose_dataset(test_df)
X_val, y_val = compose_dataset(val_df)

print('Train data shape: {}, Labels shape: {}'.format(X_train.shape, y_train.
 ↪shape))
print('Test data shape: {}, Labels shape: {}'.format(X_test.shape, y_test.
 ↪shape))
```

```
print('Validation data shape: {}, Labels shape: {}'.format(X_val.shape, y_val.
  ↪shape))
```

```
Train data shape: (106, 196, 196, 1), Labels shape: (106,)
Test data shape: (20, 196, 196, 1), Labels shape: (20,)
Validation data shape: (6, 196, 196, 1), Labels shape: (6,)
```

Dado que el dataset de imágenes es pequeño se va a hacer una aumentación sintética de los datos. Esto es generar nuevas imágenes a partir de las que ya están introducidas pero con pequeñas modificaciones, como una unos grado de rotación,zoom, algo de ruido, hacer simerias...etc

[ ]:
```
# Generador de imágenes
datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range = 0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False,
    vertical_flip=False
)

datagen.fit(X_train)#solo se aplica en el training set
```

[ ]:
```
print('Train data shape: {}, Labels shape: {}'.format(X_train.shape, y_train.
  ↪shape))
```

```
Train data shape: (106, 196, 196, 1), Labels shape: (106,)
```

En la siguiente celda,el códido convierte las etiquetas de entrenamiento, prueba y validación en representación one-hot encoding, es decir, se convierten las Ys en variables categóricas.

X representan las imágenes, con dimension 196x196 píxeles

Y representan las etiquetas, que son 0,1,2 de pequeño, mediano y grande

Categorizar las Ys produce una descomposición de más columnas, una para cada etiqueta.

[ ]:
```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_val = to_categorical(y_val)
```

#Creación de la CNN

[ ]:
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same',␣
  ↪activation='relu', input_shape=(196, 196, 1)))
#model.add(Conv2D(filters=8, kernel_size=(3,3), padding='same',␣
  ↪activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3)))
```

```
model.add(Conv2D(filters=16, kernel_size=(2,2), padding='same',␣
 ↪activation='relu'))
#model.add(Conv2D(filters=16, kernel_size=(2,2), padding='same',␣
 ↪activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(3, activation='softmax'))

optimizer = Adam(learning_rate=0.0001)
model.compile(loss='categorical_crossentropy', optimizer=optimizer,␣
 ↪metrics=['accuracy'])

callback = EarlyStopping(monitor='loss', patience=7)
history = model.fit(datagen.flow(X_train,y_train, batch_size=2),␣
 ↪validation_data=(X_test, y_test), epochs = 100, callbacks=callback, verbose␣
 ↪= 1)
```

```
Epoch 1/100
53/53 [==============================] - 5s 72ms/step - loss: 1.1191 - accuracy:
0.2830 - val_loss: 1.0955 - val_accuracy: 0.3500
Epoch 2/100
53/53 [==============================] - 4s 75ms/step - loss: 1.1013 - accuracy:
0.3113 - val_loss: 1.0832 - val_accuracy: 0.3500
Epoch 3/100
53/53 [==============================] - 3s 63ms/step - loss: 1.1094 - accuracy:
0.3019 - val_loss: 1.0782 - val_accuracy: 0.7000
Epoch 4/100
53/53 [==============================] - 5s 86ms/step - loss: 1.1014 - accuracy:
0.3396 - val_loss: 1.0768 - val_accuracy: 0.5500
Epoch 5/100
53/53 [==============================] - 4s 67ms/step - loss: 1.0921 - accuracy:
0.3585 - val_loss: 1.0682 - val_accuracy: 0.3500
Epoch 6/100
53/53 [==============================] - 3s 63ms/step - loss: 1.0925 - accuracy:
0.3585 - val_loss: 1.0683 - val_accuracy: 0.3500
Epoch 7/100
53/53 [==============================] - 3s 64ms/step - loss: 1.0880 - accuracy:
0.3679 - val_loss: 1.0601 - val_accuracy: 0.3500
Epoch 8/100
53/53 [==============================] - 5s 100ms/step - loss: 1.0845 -
accuracy: 0.4245 - val_loss: 1.0516 - val_accuracy: 0.3500
```

```
Epoch 9/100
53/53 [==============================] - 3s 61ms/step - loss: 1.0792 - accuracy:
0.3774 - val_loss: 1.0444 - val_accuracy: 0.3500
Epoch 10/100
53/53 [==============================] - 5s 89ms/step - loss: 1.0764 - accuracy:
0.3962 - val_loss: 1.0282 - val_accuracy: 0.4500
Epoch 11/100
53/53 [==============================] - 4s 71ms/step - loss: 1.0836 - accuracy:
0.3962 - val_loss: 1.0260 - val_accuracy: 0.5000
Epoch 12/100
53/53 [==============================] - 3s 61ms/step - loss: 1.0801 - accuracy:
0.4528 - val_loss: 1.0175 - val_accuracy: 0.7000
Epoch 13/100
53/53 [==============================] - 4s 72ms/step - loss: 1.0554 - accuracy:
0.5094 - val_loss: 0.9939 - val_accuracy: 0.5000
Epoch 14/100
53/53 [==============================] - 4s 71ms/step - loss: 1.0510 - accuracy:
0.5189 - val_loss: 0.9648 - val_accuracy: 0.5500
Epoch 15/100
53/53 [==============================] - 3s 62ms/step - loss: 1.0505 - accuracy:
0.4340 - val_loss: 0.9436 - val_accuracy: 0.4000
Epoch 16/100
53/53 [==============================] - 5s 96ms/step - loss: 1.0574 - accuracy:
0.3774 - val_loss: 0.9309 - val_accuracy: 0.6000
Epoch 17/100
53/53 [==============================] - 4s 68ms/step - loss: 1.0314 - accuracy:
0.4434 - val_loss: 0.9145 - val_accuracy: 0.6000
Epoch 18/100
53/53 [==============================] - 3s 60ms/step - loss: 1.0345 - accuracy:
0.5283 - val_loss: 0.8934 - val_accuracy: 0.6000
Epoch 19/100
53/53 [==============================] - 3s 64ms/step - loss: 1.0210 - accuracy:
0.4906 - val_loss: 0.8878 - val_accuracy: 0.7500
Epoch 20/100
53/53 [==============================] - 5s 100ms/step - loss: 1.0110 -
accuracy: 0.4906 - val_loss: 0.8574 - val_accuracy: 0.7500
Epoch 21/100
53/53 [==============================] - 3s 60ms/step - loss: 0.9649 - accuracy:
0.5660 - val_loss: 0.7965 - val_accuracy: 0.6500
Epoch 22/100
53/53 [==============================] - 5s 99ms/step - loss: 0.9993 - accuracy:
0.5094 - val_loss: 0.7895 - val_accuracy: 0.8000
Epoch 23/100
53/53 [==============================] - 3s 63ms/step - loss: 0.9865 - accuracy:
0.5000 - val_loss: 0.7823 - val_accuracy: 0.7500
Epoch 24/100
53/53 [==============================] - 5s 91ms/step - loss: 0.9736 - accuracy:
0.5094 - val_loss: 0.7705 - val_accuracy: 0.8000
```

```
Epoch 25/100
53/53 [==============================] - 3s 61ms/step - loss: 0.9237 - accuracy:
0.5472 - val_loss: 0.7458 - val_accuracy: 0.8500
Epoch 26/100
53/53 [==============================] - 3s 63ms/step - loss: 0.9096 - accuracy:
0.6132 - val_loss: 0.7377 - val_accuracy: 0.7000
Epoch 27/100
53/53 [==============================] - 4s 83ms/step - loss: 0.9236 - accuracy:
0.5566 - val_loss: 0.7826 - val_accuracy: 0.7500
Epoch 28/100
53/53 [==============================] - 3s 64ms/step - loss: 0.9165 - accuracy:
0.5755 - val_loss: 0.7297 - val_accuracy: 0.7000
Epoch 29/100
53/53 [==============================] - 4s 72ms/step - loss: 0.9270 - accuracy:
0.5849 - val_loss: 0.6792 - val_accuracy: 0.7500
Epoch 30/100
53/53 [==============================] - 4s 71ms/step - loss: 0.8566 - accuracy:
0.6132 - val_loss: 0.6455 - val_accuracy: 0.7500
Epoch 31/100
53/53 [==============================] - 3s 64ms/step - loss: 0.9367 - accuracy:
0.5660 - val_loss: 0.6305 - val_accuracy: 0.9000
Epoch 32/100
53/53 [==============================] - 5s 93ms/step - loss: 0.8538 - accuracy:
0.6981 - val_loss: 0.5554 - val_accuracy: 0.9000
Epoch 33/100
53/53 [==============================] - 4s 69ms/step - loss: 0.8328 - accuracy:
0.6415 - val_loss: 0.6990 - val_accuracy: 0.7500
Epoch 34/100
53/53 [==============================] - 3s 64ms/step - loss: 0.8607 - accuracy:
0.6604 - val_loss: 0.5539 - val_accuracy: 0.8500
Epoch 35/100
53/53 [==============================] - 3s 65ms/step - loss: 0.8895 - accuracy:
0.5660 - val_loss: 0.6036 - val_accuracy: 0.8000
Epoch 36/100
53/53 [==============================] - 4s 76ms/step - loss: 0.8640 - accuracy:
0.6604 - val_loss: 0.6545 - val_accuracy: 0.8000
Epoch 37/100
53/53 [==============================] - 3s 64ms/step - loss: 0.8637 - accuracy:
0.5849 - val_loss: 0.5185 - val_accuracy: 0.9000
Epoch 38/100
53/53 [==============================] - 4s 68ms/step - loss: 0.8347 - accuracy:
0.6415 - val_loss: 0.5262 - val_accuracy: 0.9000
Epoch 39/100
53/53 [==============================] - 4s 72ms/step - loss: 0.8297 - accuracy:
0.6038 - val_loss: 0.6180 - val_accuracy: 0.8000
Epoch 40/100
53/53 [==============================] - 3s 63ms/step - loss: 0.8276 - accuracy:
0.6792 - val_loss: 0.5627 - val_accuracy: 0.8000
```

```
Epoch 41/100
53/53 [==============================] - 4s 68ms/step - loss: 0.8032 - accuracy:
0.6604 - val_loss: 0.4917 - val_accuracy: 0.9000
Epoch 42/100
53/53 [==============================] - 4s 68ms/step - loss: 0.8628 - accuracy:
0.6604 - val_loss: 0.5197 - val_accuracy: 0.8500
Epoch 43/100
53/53 [==============================] - 3s 65ms/step - loss: 0.7861 - accuracy:
0.6321 - val_loss: 0.5292 - val_accuracy: 0.8000
Epoch 44/100
53/53 [==============================] - 5s 94ms/step - loss: 0.7970 - accuracy:
0.6321 - val_loss: 0.5443 - val_accuracy: 0.8000
Epoch 45/100
53/53 [==============================] - 3s 64ms/step - loss: 0.8000 - accuracy:
0.6038 - val_loss: 0.5953 - val_accuracy: 0.8000
Epoch 46/100
53/53 [==============================] - 3s 64ms/step - loss: 0.7743 - accuracy:
0.6509 - val_loss: 0.5220 - val_accuracy: 0.8000
Epoch 47/100
53/53 [==============================] - 4s 78ms/step - loss: 0.7864 - accuracy:
0.6604 - val_loss: 0.5163 - val_accuracy: 0.8000
Epoch 48/100
53/53 [==============================] - 4s 69ms/step - loss: 0.8142 - accuracy:
0.6226 - val_loss: 0.5013 - val_accuracy: 0.9000
Epoch 49/100
53/53 [==============================] - 3s 61ms/step - loss: 0.7853 - accuracy:
0.6226 - val_loss: 0.5473 - val_accuracy: 0.8000
Epoch 50/100
53/53 [==============================] - 3s 61ms/step - loss: 0.7993 - accuracy:
0.6792 - val_loss: 0.4830 - val_accuracy: 0.8000
Epoch 51/100
53/53 [==============================] - 4s 84ms/step - loss: 0.7633 - accuracy:
0.6415 - val_loss: 0.5448 - val_accuracy: 0.8000
Epoch 52/100
53/53 [==============================] - 3s 62ms/step - loss: 0.8326 - accuracy:
0.6321 - val_loss: 0.3953 - val_accuracy: 0.9500
Epoch 53/100
53/53 [==============================] - 4s 74ms/step - loss: 0.7905 - accuracy:
0.6604 - val_loss: 0.5418 - val_accuracy: 0.8000
Epoch 54/100
53/53 [==============================] - 4s 68ms/step - loss: 0.8344 - accuracy:
0.5849 - val_loss: 0.4682 - val_accuracy: 0.9500
Epoch 55/100
53/53 [==============================] - 3s 62ms/step - loss: 0.7736 - accuracy:
0.6509 - val_loss: 0.5063 - val_accuracy: 0.8000
Epoch 56/100
53/53 [==============================] - 3s 61ms/step - loss: 0.8049 - accuracy:
0.6792 - val_loss: 0.4988 - val_accuracy: 0.8000
```

```
Epoch 57/100
53/53 [==============================] - 5s 95ms/step - loss: 0.7530 - accuracy:
0.6509 - val_loss: 0.5415 - val_accuracy: 0.8000
Epoch 58/100
53/53 [==============================] - 4s 68ms/step - loss: 0.7649 - accuracy:
0.6509 - val_loss: 0.5270 - val_accuracy: 0.8000
Epoch 59/100
53/53 [==============================] - 3s 61ms/step - loss: 0.8669 - accuracy:
0.6132 - val_loss: 0.4991 - val_accuracy: 0.9000
Epoch 60/100
53/53 [==============================] - 4s 81ms/step - loss: 0.8170 - accuracy:
0.5566 - val_loss: 0.5314 - val_accuracy: 0.8000
Epoch 61/100
53/53 [==============================] - 4s 69ms/step - loss: 0.8077 - accuracy:
0.6604 - val_loss: 0.5160 - val_accuracy: 0.8000
Epoch 62/100
53/53 [==============================] - 3s 64ms/step - loss: 0.7386 - accuracy:
0.7075 - val_loss: 0.4777 - val_accuracy: 0.8500
Epoch 63/100
53/53 [==============================] - 4s 83ms/step - loss: 0.7618 - accuracy:
0.6415 - val_loss: 0.5271 - val_accuracy: 0.8000
Epoch 64/100
53/53 [==============================] - 4s 80ms/step - loss: 0.7802 - accuracy:
0.6792 - val_loss: 0.4989 - val_accuracy: 0.8000
Epoch 65/100
53/53 [==============================] - 3s 64ms/step - loss: 0.7189 - accuracy:
0.6792 - val_loss: 0.4248 - val_accuracy: 0.9000
Epoch 66/100
53/53 [==============================] - 5s 94ms/step - loss: 0.8065 - accuracy:
0.6604 - val_loss: 0.4714 - val_accuracy: 0.8000
Epoch 67/100
53/53 [==============================] - 6s 112ms/step - loss: 0.7489 -
accuracy: 0.6981 - val_loss: 0.4426 - val_accuracy: 0.8000
Epoch 68/100
53/53 [==============================] - 3s 62ms/step - loss: 0.7327 - accuracy:
0.6604 - val_loss: 0.4310 - val_accuracy: 0.8000
Epoch 69/100
53/53 [==============================] - 3s 65ms/step - loss: 0.7216 - accuracy:
0.6698 - val_loss: 0.4696 - val_accuracy: 0.8000
Epoch 70/100
53/53 [==============================] - 5s 89ms/step - loss: 0.7002 - accuracy:
0.7075 - val_loss: 0.3978 - val_accuracy: 0.9000
Epoch 71/100
53/53 [==============================] - 4s 71ms/step - loss: 0.7585 - accuracy:
0.6415 - val_loss: 0.5039 - val_accuracy: 0.8000
Epoch 72/100
53/53 [==============================] - 3s 61ms/step - loss: 0.7185 - accuracy:
0.6604 - val_loss: 0.3879 - val_accuracy: 0.9500
```

```
Epoch 73/100
53/53 [==============================] - 4s 71ms/step - loss: 0.7264 - accuracy:
0.6981 - val_loss: 0.5894 - val_accuracy: 0.8000
Epoch 74/100
53/53 [==============================] - 4s 72ms/step - loss: 0.7961 - accuracy:
0.6226 - val_loss: 0.4288 - val_accuracy: 0.9000
Epoch 75/100
53/53 [==============================] - 3s 65ms/step - loss: 0.7223 - accuracy:
0.6604 - val_loss: 0.4002 - val_accuracy: 0.9000
Epoch 76/100
53/53 [==============================] - 5s 93ms/step - loss: 0.6955 - accuracy:
0.6981 - val_loss: 0.3792 - val_accuracy: 0.9000
Epoch 77/100
53/53 [==============================] - 4s 66ms/step - loss: 0.6583 - accuracy:
0.7170 - val_loss: 0.4904 - val_accuracy: 0.8500
Epoch 78/100
53/53 [==============================] - 3s 64ms/step - loss: 0.7667 - accuracy:
0.6698 - val_loss: 0.4386 - val_accuracy: 0.8500
Epoch 79/100
53/53 [==============================] - 5s 99ms/step - loss: 0.6597 - accuracy:
0.6981 - val_loss: 0.4632 - val_accuracy: 0.8500
Epoch 80/100
53/53 [==============================] - 3s 65ms/step - loss: 0.7378 - accuracy:
0.6792 - val_loss: 0.4390 - val_accuracy: 0.8500
Epoch 81/100
53/53 [==============================] - 5s 101ms/step - loss: 0.6795 -
accuracy: 0.6887 - val_loss: 0.3460 - val_accuracy: 0.9500
Epoch 82/100
53/53 [==============================] - 3s 62ms/step - loss: 0.6150 - accuracy:
0.7358 - val_loss: 0.4005 - val_accuracy: 0.9000
Epoch 83/100
53/53 [==============================] - 5s 95ms/step - loss: 0.6993 - accuracy:
0.7075 - val_loss: 0.5390 - val_accuracy: 0.8000
Epoch 84/100
53/53 [==============================] - 3s 62ms/step - loss: 0.7423 - accuracy:
0.6604 - val_loss: 0.4078 - val_accuracy: 0.9000
Epoch 85/100
53/53 [==============================] - 3s 62ms/step - loss: 0.6756 - accuracy:
0.7264 - val_loss: 0.4172 - val_accuracy: 0.8500
Epoch 86/100
53/53 [==============================] - 5s 86ms/step - loss: 0.7350 - accuracy:
0.6226 - val_loss: 0.4331 - val_accuracy: 0.9000
Epoch 87/100
53/53 [==============================] - 3s 63ms/step - loss: 0.7533 - accuracy:
0.7075 - val_loss: 0.3812 - val_accuracy: 0.9500
Epoch 88/100
53/53 [==============================] - 3s 65ms/step - loss: 0.6590 - accuracy:
0.7075 - val_loss: 0.3794 - val_accuracy: 0.9500
```

```
Epoch 89/100
53/53 [==============================] - 5s 92ms/step - loss: 0.6729 - accuracy:
0.6792 - val_loss: 0.3526 - val_accuracy: 0.9500
```

[ ]:
```
model.summary() #Resumen de la arquitetura
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 196, 196, 32)      320

 max_pooling2d_4 (MaxPooling  (None, 65, 65, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 65, 65, 16)        2064

 max_pooling2d_5 (MaxPooling  (None, 32, 32, 16)       0
 2D)

 flatten_2 (Flatten)         (None, 16384)             0

 dense_4 (Dense)             (None, 32)                524320

 dropout_2 (Dropout)         (None, 32)                0

 dense_5 (Dense)             (None, 3)                 99

=================================================================
Total params: 526,803
Trainable params: 526,803
Non-trainable params: 0
_________________________________________________________________
```

4 Save model

[ ]:
```
!ls
```

```
Calabacines  gdrive  my_model.h5  sample_data
```

[ ]:
```
# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('my_model.h5')
```

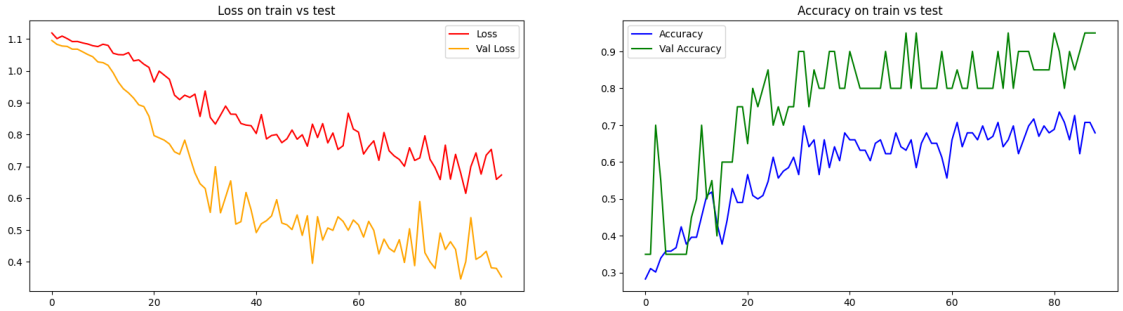
5 Evaluación

Las siguientes celdillas van a ser de evaluación de las pérdidas y cálculo de las métricas que puedan dar una idea de la calidad del entrenamiento.

En este caso vamos a representar 2 gráficas con 2 curvas: - Las pérdidas, que indican como de equivocado está el modelo en cada interación - Y la accuracy, la métrica que se ha elegido para evaluar la calidad del training. Esta es la que se suele usar para labores de clasificación con imágenes.

A continuación, las gráficas posteriores se podrá apreciar una leyenda donde los párametros (val) hacen referencia al subdataset test

[ ]:
```
plt.figure(figsize=(20,5))

# plot loss & val loss
plt.subplot(1,2,1)
sns.lineplot(x=history.epoch, y=history.history['loss'], color='red',␣
 ↪label='Loss')
sns.lineplot(x=history.epoch, y=history.history['val_loss'], color='orange',␣
 ↪label='Val Loss')
plt.title('Loss on train vs test')
plt.legend(loc='best')

# plot accuracy and val accuracy
plt.subplot(1,2,2)
sns.lineplot(x=history.epoch, y=history.history['accuracy'], color='blue',␣
 ↪label='Accuracy')
sns.lineplot(x=history.epoch, y=history.history['val_accuracy'], color='green',␣
 ↪label='Val Accuracy')
plt.title('Accuracy on train vs test')
plt.legend(loc='best')

plt.show()
```



[ ]:
```
score = model.evaluate(X_test, y_test, verbose = 0) #Se evalua cuántas imágenes␣
 ↪de la subdivisión test es capaz de predecir bien (calculando las pérdidas y␣
 ↪el accuracy)

print(f"Test Loss: {score[0]}")
print(f"Test Accuracy: {score[1] * 100:.2f}%")
```

```
Test Loss: 0.3526293635368347
Test Accuracy: 95.00%
```

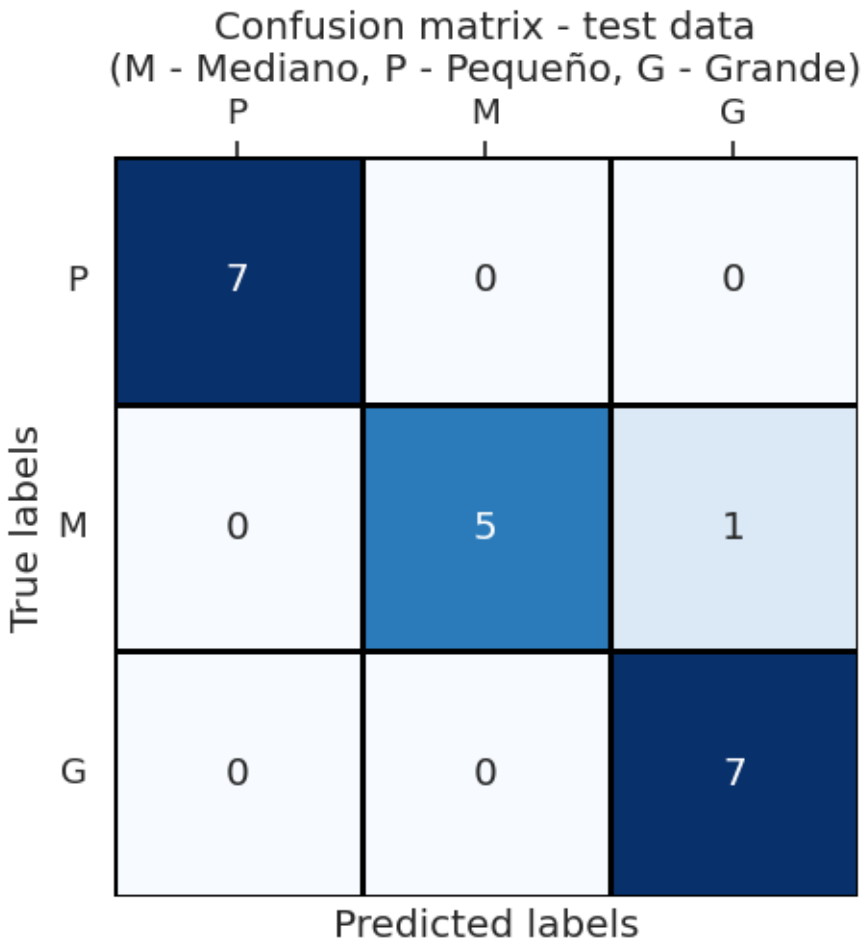
[ ]:
```
score2 = model.evaluate(X_train, y_train, verbose = 0)  #Se evalua cuantas␣
 ↪imagenes de la subdivisón train es capaz de predecir bien
print(f"Train Loss: {score2[0]}")
print(f"Train Accuracy: {score2[1] * 100:.2f}%")
```

```
Train Loss: 0.5105329751968384
Train Accuracy: 80.19%
```

[ ]:
```
y_test_hat = model.predict(X_test, batch_size=4) #Se utiliza el modelo␣
 ↪entrenado para realizar predicciones en los datos de prueba
y_test_hat = np.argmax(y_test_hat, axis=1) #Se obtiene las predicciones de␣
 ↪clase correspondientes a las probabilidades predichas
y_test = np.argmax(y_test, axis=1) #Se convierte la representación one-hot␣
 ↪encoding de las etiquetas en su forma de índice de clase
```

```
5/5 [==============================] - 0s 49ms/step
```

[ ]:
```
conf_m = confusion_matrix(y_test, y_test_hat) #Se calcula la matriz de confusión
clas_r = classification_report(y_test, y_test_hat) #Se genera un informe de␣
 ↪clasificación

plt.figure(figsize=(5,5))
sns.set(font_scale=1.2)
ax = sns.heatmap(conf_m, annot=True,xticklabels=['P','M', 'G'],␣
 ↪yticklabels=['P','M', 'G'], cbar=False, cmap='Blues',linewidths=1,␣
 ↪linecolor='black', fmt='.0f')
plt.yticks(rotation=0)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
ax.xaxis.set_ticks_position('top')
plt.title('Confusion matrix - test data\n(M - Mediano, P - Pequeño, G -␣
 ↪Grande)')
plt.show()

print('Classification report on test data')
print(clas_r)
```




```
Classification report on test data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      0.83      0.91         6
           2       0.88      1.00      0.93         7

    accuracy                           0.95        20
   macro avg       0.96      0.94      0.95        20
weighted avg       0.96      0.95      0.95        20
```

5.1 Validación

[ ]:
```
y_val_hat = model.predict(X_val, batch_size=4)
y_val_hat = np.argmax(y_val_hat, axis=1)
y_val = np.argmax(y_val, axis=1)
```

```
2/2 [==============================] - 0s 43ms/step
```

Integración CNN + Robot

May 27, 2023

[ ]:
```
# Commented out IPython magic to ensure Python compatibility.
import glob
import random as rn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop, Adam
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix


# %matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import sys
import math
import time
import datetime
import random
import traceback
import threading


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop, Adam
```

```
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix

i=0
j=0

#Encendido del robot
"""
# xArm-Python-SDK: https://github.com/xArm-Developer/xArm-Python-SDK
# git clone git@github.com:xArm-Developer/xArm-Python-SDK.git
# cd xArm-Python-SDK
# python setup.py install
"""
try:
    from xarm.tools import utils
except:
    pass
from xarm import version
from xarm.wrapper import XArmAPI

def pprint(*args, **kwargs):
    try:
        stack_tuple = traceback.extract_stack(limit=2)[0]
        print('[{}][{}] {}'.format(time.strftime('%Y-%m-%d %H:%M:%S', time.
 ↪localtime(time.time())), stack_tuple[1], ' '.join(map(str, args))))
    except:
        print(*args, **kwargs)

pprint('xArm-Python-SDK Version:{}'.format(version.__version__))

arm = XArmAPI('192.168.1.221')
arm.clean_warn()
arm.clean_error()
arm.motion_enable(True)
arm.set_mode(0)
arm.set_state(0)
time.sleep(1)

variables = {}
params = {'speed': 100, 'acc': 2000, 'angle_speed': 20, 'angle_acc': 500,␣
 ↪'events': {}, 'variables': variables, 'callback_in_thread': True, 'quit':␣
 ↪False}

def create_folder():
    # Nombre de la carpeta con la fecha y hora actual
    fecha_actual = time.strftime("%d-%m-%Y-")
    hora_actual = time.strftime("%H_%M_%S")
```

```
    folder_name = "Data_" + fecha_actual + hora_actual

    # Ruta de la carpeta
    folder_path = os.path.join(os.getcwd(), folder_name)

    # Crear la carpeta si no existe
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)

    return folder_path

def delete_folder(folder_path):
    # Eliminar la carpeta y su contenido
    if os.path.exists(folder_path):
        for file_name in os.listdir(folder_path):
            file_path = os.path.join(folder_path, file_name)
            os.remove(file_path)
        os.rmdir(folder_path)

def take_photo(folder_path):
    # Abrir la cámara
    cap = cv2.VideoCapture(1)

    # Comprobar si la cámara se abrió correctamente
    if not cap.isOpened():
        print("No se pudo abrir la cámara")
        return

    # Leer un fotograma de la cámara
    ret, frame = cap.read()

    # Comprobar si se capturó correctamente el fotograma
    if not ret:
        print("No se pudo capturar la foto")
        return

    # Guardar la foto en la carpeta
    photo_path = os.path.join(folder_path, "captura.jpg")
    cv2.imwrite(photo_path, frame)

    # Cerrar la cámara
    cap.release()

    # Devolver el directorio de la foto capturada
    return photo_path
```

```
# Register error/warn changed callback
def error_warn_change_callback(data):
    if data and data['error_code'] != 0:
        params['quit'] = True
        pprint('err={}, quit'.format(data['error_code']))
        arm.release_error_warn_changed_callback(error_warn_change_callback)
arm.register_error_warn_changed_callback(error_warn_change_callback)


# Register state changed callback
def state_changed_callback(data):
    if data and data['state'] == 4:
        if arm.version_number[0] >= 1 and arm.version_number[1] >= 1 and arm.
 ↪version_number[2] > 0:
            params['quit'] = True
            pprint('state=4, quit')
            arm.release_state_changed_callback(state_changed_callback)
arm.register_state_changed_callback(state_changed_callback)


# Register counter value changed callback
if hasattr(arm, 'register_count_changed_callback'):
    def count_changed_callback(data):
        if not params['quit']:
            pprint('counter val: {}'.format(data['count']))
    arm.register_count_changed_callback(count_changed_callback)


# Register connect changed callback
def connect_changed_callback(data):
    if data and not data['connected']:
        params['quit'] = True
        pprint('disconnect, connected={}, reported={}, quit'.
 ↪format(data['connected'], data['reported']))
        arm.release_connect_changed_callback(error_warn_change_callback)
arm.register_connect_changed_callback(connect_changed_callback)

if not params['quit']:
    params['speed'] = 1000


# %matplotlib inline
import warnings
warnings.filterwarnings('ignore')

# Capturar una foto y guardarla en la carpeta
#foto_directorio = take_photo(folder_path)
```

```
#print("Foto capturada:", foto_directorio)


# Preprocesado de datos

#El preprocesado de las imágenes,es necesario para adaptar la información al␣
 ↪máximo para facilitarle el entrenamiento y aprendizaje a la red neuronal.



def process_data(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (196, 196))#cambiamos el tamaño (los píxeles) de las␣
 ↪imagenes para la entrada de datos
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)#a blanco y negro, facilita la␣
 ↪detección de patrones y aprendizaje
    img = img/255.0#normalización de la imagen, 255 es la gama de colores rgb
    img = np.reshape(img, (196,196,1))

    return img

def compose_dataset(df):#asigna las imagenes tratadas de nuevo a susetiquetas (␣
 ↪la etiqueta es la clasificación)
    data = []
    labels = []

    for img_path, label in df.values:
        data.append(process_data(img_path))
        labels.append(label)

    return np.array(data), np.array(labels)


import tensorflow as tf

# Recreate the exact same model, including its weights and the optimizer
model = tf.keras.models.load_model('my_model.h5')

# Show the model architecture
model.summary()

#Movimientos de inicio del robot para tomar la foto
#remark

if arm.error_code == 0 and not params['quit']:
    code = arm.set_gripper_position(850, wait=True, speed=5000,␣
 ↪auto_enable=True)
```

```
    if code != 0:
        params['quit'] = True
        pprint('set_gripper_position, code={}'.format(code))
if arm.error_code == 0 and not params['quit']:
    code = arm.set_position(*[152.7, 0.6, 704.7, 180.0, 0.0, -90.0],␣
 ↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
    if code != 0:
        params['quit'] = True
        pprint('set_position, code={}'.format(code))


while j<7:

    # Crear una carpeta
    folder_path = create_folder()
    print("Carpeta creada:", folder_path)


    ret, frame = cap.read()#Capturacion de imagen
    Folder=os.path.join(folder_path, "Calabacin_{}.png".format(id))
    cv2.imwrite(Folder,frame) # Imagen guardada en dicha carpeta

    test_cases_2 = folder_path + 'Calabacin_{}.png'.format(id) #Direccion donde␣
 ↪esta guardada la imagen recien hecha
    X_val=process_data(test_cases_2)


    y_val_hat = model.predict(X_val, batch_size=1)
    y_val_hat = np.argmax(y_val_hat, axis=1)

    #El brazo realiza el movimiento de bajar y abrir el griper

    if arm.error_code == 0 and not params['quit']:
        code = arm.set_position(*[152.7, 0.6, 240.2, 180.0, 0.0, -90.0],␣
 ↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
        if code != 0:
            params['quit'] = True
            pprint('set_position, code={}'.format(code))
    if arm.error_code == 0 and not params['quit']:
        code = arm.set_gripper_position(109, wait=True, speed=5000,␣
 ↪auto_enable=True)
        if code != 0:
            params['quit'] = True
            pprint('set_gripper_position, code={}'.format(code))


    if y_val_hat[i]==0:  "Detección Pequeños"
```

```
      if arm.error_code == 0 and not params['quit']:
          code = arm.set_position(*[152.7, 0.6, 558.0, -180.0, 0.0, -90.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
          if code != 0:
              params['quit'] = True
              pprint('set_position, code={}'.format(code))
      if arm.error_code == 0 and not params['quit']:
          code = arm.set_position(*[425.4, 15.6, 554.4, -180.0, 0.0, 1.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
          if code != 0:
              params['quit'] = True
              pprint('set_position, code={}'.format(code))
      if arm.error_code == 0 and not params['quit']:
          code = arm.set_position(*[425.4, 15.6, 240.7, -180.0, 0.0, 1.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
          if code != 0:
              params['quit'] = True
              pprint('set_position, code={}'.format(code))
      if arm.error_code == 0 and not params['quit']:
          code = arm.set_gripper_position(850, wait=True, speed=5000,␣
↪auto_enable=True)
          if code != 0:
              params['quit'] = True
              pprint('set_gripper_position, code={}'.format(code))
      if arm.error_code == 0 and not params['quit']:
          code = arm.set_position(*[425.1, 15.6, 483.6, -180.0, 0.0, 1.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
          if code != 0:
              params['quit'] = True
              pprint('set_position, code={}'.format(code))
      if arm.error_code == 0 and not params['quit']:
          code = arm.set_position(*[152.7, 0.6, 704.7, 180.0, 0.0, -90.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
          if code != 0:
              params['quit'] = True
              pprint('set_position, code={}'.format(code))



  if y_val_hat[i]==1 : "Detección Medianos"
      if arm.error_code == 0 and not params['quit']:
          code = arm.set_position(*[152.7, 0.6, 558.0, -180.0, 0.0, -90.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
          if code != 0:
```

```
                params['quit'] = True
                pprint('set_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_position(*[425.4, 15.6, 554.4, -180.0, 0.0, 1.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
            if code != 0:
                params['quit'] = True
                pprint('set_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_position(*[425.4, 161.7, 240.7, 180.0, 0.0, 1.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
            if code != 0:
                params['quit'] = True
                pprint('set_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_gripper_position(850, wait=True, speed=5000,␣
↪auto_enable=True)
            if code != 0:
                params['quit'] = True
                pprint('set_gripper_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_position(*[425.1, 15.6, 483.6, -180.0, 0.0, 1.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
            if code != 0:
                params['quit'] = True
                pprint('set_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_position(*[152.7, 0.6, 704.7, 180.0, 0.0, -90.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
            if code != 0:
                params['quit'] = True
                pprint('set_position, code={}'.format(code))



   if y_val_hat[i]==2 : "Detección Grandes"
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_position(*[152.7, 0.6, 558.0, -180.0, 0.0, -90.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
            if code != 0:
                params['quit'] = True
                pprint('set_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_position(*[425.4, 15.6, 554.4, -180.0, 0.0, 1.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
            if code != 0:
```

```
                params['quit'] = True
                pprint('set_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_position(*[425.4, 345.3, 240.7, 180.0, 0.0, 1.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
            if code != 0:
                params['quit'] = True
                pprint('set_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_gripper_position(850, wait=True, speed=5000,␣
↪auto_enable=True)
            if code != 0:
                params['quit'] = True
                pprint('set_gripper_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_position(*[425.1, 15.6, 483.6, -180.0, 0.0, 1.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
            if code != 0:
                params['quit'] = True
                pprint('set_position, code={}'.format(code))
        if arm.error_code == 0 and not params['quit']:
            code = arm.set_position(*[152.7, 0.6, 704.7, 180.0, 0.0, -90.0],␣
↪speed=params['speed'], mvacc=params['acc'], radius=-1.0, wait=True)
            if code != 0:
                params['quit'] = True
                pprint('set_position, code={}'.format(code))
    #remark
    j++
    # Eliminar la carpeta y su contenido
    delete_folder(folder_path)
    print("Carpeta eliminada:", folder_path)
    #remark
```