ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

APLICACIONES DE MACHINE LEARNING PARA EL ANÁLISIS DE COMIDA

OBTENCIÓN Y DEPURACIÓN DE LOS DATOS PARA EL ANÁLISIS AUTOMÁTICO DE COMIDA

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

DAVID ALEXANDER SANTOS MOROCHO
david.santos@epn.edu.ec

DIRECTOR: MSC. FRANKLIN LEONEL SANCHEZ CATOTA
franklin.sanchez@epn.edu.ec

DMQ, febrero 2024

CERTIFICACIONES

Yo, DAVID ALEXANDER SANTOS MOROCHO declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

DAVID ALEXANDER SANTOS MOROCHO

Certifico que el presente trabajo de integración curricular fue desarrollado por DAVID ALEXANDER SANTOS MOROCHO, bajo mi supervisión.

MSc. FRANKLIN LEONEL SANCHEZ CATOTA
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

DAVID ALEXANDER SANTOS MOROCHO
MSc. FRANKLIN LEONEL SANCHEZ CATOTA
PhD. CHRISTIAN JOSÉ TIPANTUÑA TENELEMA

DEDICATORIA

A mis padres y a mi querido hermano por su amor y apoyo incondicional, que han sido un pilar fundamental en cada fase de mi trayectoria académica. Su constante respaldo ha sido invaluable, brindándome fuerza y estabilidad en momentos clave.

A toda mi familia, a quienes considero un honor dedicar este trabajo para que sea un ejemplo inspirador no solo para mis primos, sino también para todos mis parientes.

A todas las personas que, de diferentes maneras, me han ofrecido valiosos consejos y orientación, dejando una huella imborrable en mi corazón.

AGRADECIMIENTO

Expreso mi profundo agradecimiento a Dios por su amor infinito y por permanecer a mi lado en cada etapa de mi vida, brindándome su constante compañía y apoyo incondicional.

A mis padres, Victoria y Bolívar. Su amor incondicional ha sido el motor que ha impulsado cada uno de mis esfuerzos y el faro que ha iluminado mi camino hacia la culminación de esta importante etapa en mi vida académica. Este logro no solo representa un hito académico, sino también un triunfo que refleja la dedicación y el sacrificio que han demostrado a lo largo de los años. Sin su apoyo inquebrantable, este camino habría sido mucho más difícil.

A mi querido hermano Erick, quien sin duda ha sido mi principal fuente de inspiración y guía a lo largo de toda mi trayectoria académica. Su apoyo incondicional ha sido fundamental en cada paso que he dado, desde los desafíos más pequeños hasta los logros más significativos.

A Steven, Paola, Jean Pierre y a todos mis amigos su apoyo incondicional ha sido crucial en cada etapa de este proceso, desde la incertidumbre hasta los logros. Son más que compañeros, han enriquecido mi experiencia con momentos inolvidables y han sido un apoyo invaluable en los momentos difíciles. Su amistad es un regalo preciado que valoro profundamente en este viaje de crecimiento personal.

Al PhD. Christian Tipantuña y al Msc. Franklin Sánchez, les expreso mi más sincero agradecimiento por su confianza, paciencia y valiosa asistencia durante el desarrollo de este proyecto.

Finalmente, a la Escuela Politécnica Nacional por las oportunidades que han enriquecido mi desarrollo académico, profesional y personal. Su entorno estimulante ha potenciado mi crecimiento en diversas áreas, desde la educación de calidad hasta los valores impartidos como la perseverancia y el trabajo en equipo, fundamentales para mi crecimiento personal.

ÍNDICE DE CONTENIDO

RESUMEN

La alimentación constituye un pilar esencial en la vida humana y su análisis puede ofrecer valiosa información sobre la salud y el bienestar de las personas. Sin embargo, este análisis es un procedimiento complejo que demanda tiempo y dedicación, lo que ha impulsado la creación de herramientas automatizadas para agilizar dicho proceso.

En el presente Trabajo de Integración Curricular, se presenta el desarrollo de una base de datos diseñada para ser compatible con algoritmos de entrenamiento y clasificación de imágenes de comida. Este documento detalla el proceso de construcción de dicho conjunto de datos, el cual se deriva de una base de datos preexistente. La obtención y procesamiento de las imágenes de comida involucra la revisión de repositorios y la consideración de parámetros de calidad y variedad en las imágenes. Asimismo, el proceso de limpieza de datos comprende la eliminación de valores atípicos, errores y duplicados, seguido de su respectivo etiquetado. Se ofrece una descripción minuciosa del proceso de segmentación y etiquetado de las imágenes.

Finalmente, se presenta la estructura definitiva de la base de datos junto con los resultados obtenidos en las pruebas de predicción con diferentes algoritmos. Se concluye que la base de datos generada es apta para su implementación en modelos de aprendizaje automático, y los resultados de las pruebas de predicción son alentadores. Este trabajo sienta las bases para futuras investigaciones en el análisis automatizado de alimentos mediante la aplicación de técnicas de Machine Learning.

PALABRAS CLAVE: Machine Learning (ML), análisis de comida, procesamiento de imágenes, base de datos, depuración de datos, segementación, etiquetamiento, SAM, reconocimiento de patrones, algoritmos de clasificación.

ABSTRACT

Diet constitutes an essential pillar in human life, and its analysis can offer valuable information about people's health and well-being. However, this analysis is a complex procedure that demands time and dedication, which has driven the creation of automated tools to streamline this process.

This Curricular Integration Project presents the development of a database designed to be compatible with food image training and classification algorithms. This document details the process of constructing this dataset derived from an existing database. The acquisition and processing of food images involve reviewing repositories and considering parameters of quality and variety in the images. Likewise, the data cleaning process removes outliers, errors, and duplicates, followed by their respective labeling. A thorough description of the image segmentation and labeling process is provided.

Finally, the definitive structure of the database is presented along with the results obtained in prediction tests with different algorithms. It is concluded that the generated database is suitable for implementation in machine learning models, and the results of the prediction tests are encouraging. This work lays the foundation for future research in automated food analysis by applying machine learning techniques.

KEYWORDS: Machine Learning (ML), food analysis, image processing, database, data cleansing, segmentation, labeling, SAM, pattern recognition, classification algorithms.

1 INTRODUCCIÓN

El aprendizaje automático (ML: Machine Learning) ha revolucionado el campo de la inteligencia artificial al permitir que las máquinas aprendan y mejoren de forma autónoma a partir de los datos proporcionados. Este avance ha brindado la capacidad de detectar patrones y tendencias en los datos, lo que habilita a las máquinas para realizar predicciones precisas y valiosas. Sin embargo, para que los modelos de ML sean efectivos, es esencial contar con datos relevantes y de alta calidad [1].

El proceso de obtención y depuración de datos desempeñan un papel crítico en la capacidad de los modelos de ML. La obtención de datos implica la recopilación de información pertinente y útil para el análisis, mientras que la depuración de datos se refiere a la limpieza y transformación de los datos de entrada para garantizar su idoneidad en el entrenamiento de modelos [1].

Por otro lado la comida es una parte fundamental de la experiencia humana y, en la era digital, su presencia en la web y en la vida diaria se ha vuelto más significativa que nunca. Es por esto que uno de las grandes aplicaciones del ML es la predicción de los alimentos para que las personas puedan hacer seguimiento de su consumo de alimentos y a tomar conciencia de su dieta diaria mediante el control de sus hábitos alimentarios. Y en los últimos años ciertos trabajos de investigación han demostrado que las técnicas de ML pueden ayudar a crear sistemas que reconozcan automáticamente diversos alimentos y estimen su cantidad. Sin embargo, una evaluación comparativa justa de estos sistemas requieren la disponibilidad de conjuntos de datos adecuados que planteen realmente los retos de la tarea de reconocimiento de alimentos en entornos sin restricciones [2]. En la actualidad, las imágenes de alimentos se comparten ampliamente en redes sociales y repositorios especializados como Kaggle, convirtiéndose en una fuente inagotable de inspiración culinaria. En este contexto, se abre un extenso horizonte de oportunidades para el ML. Un ejemplo concreto de esta posibilidad es el desarrollo de una aplicación que, a partir de una fotografía del plato de comida, pueda prever la cantidad de calorías que se consumirán.

En este sentido el presente Trabajo de Integración Curricular se enfoca en el desafío de obtener y depurar datos de imágenes relacionadas con alimentos. Este proceso implica la recopilación de información sobre imágenes de comida y la eliminación de cualquier dato innecesario o redundante que pueda afectar la calidad de los datos.

A lo largo de este trabajo, se exploran a fondo los aspectos cruciales de la obtención y

depuración de datos en el contexto de la inteligencia artificial y el ML, centrándose específicamente en la tarea de recopilar y mejorar la información relacionada con imágenes de alimentos. Este trabajo es esencial para garantizar que los modelos de ML que se desarrolle posteriormente pueda realizar predicciones precisas y útiles.

1.1 OBJETIVO GENERAL

Generar un conjunto de datos de imágenes de comida limpio y etiquetado para su posible uso en modelos de Machine Learning.

1.2 OBJETIVOS ESPECÍFICOS

1. Estudiar el fundamento teórico asociado al Machine Learning, incluyendo la adquisición y depuración de datos.
2. Describir el proceso de obtención y procesamiento de imágenes de comida.
3. Obtener imágenes de comida mediante la revisión de repositorios, considerando parámetros de calidad y variedad en las imágenes.
4. Procesar el conjunto de datos obtenidos para, limpiarlos incluyendo la identificación y eliminación de valores atípicos y errores, así como datos duplicados; y sus respectivo etiquetado.

1.3 ALCANCE

El presente trabajo de integración curricular tiene por objetivo realizar el proceso de adquisición y limpieza de datos correspondientes a imágenes de comida con la finalidad que estos datos puedan servir para el proceso de entrenamiento y clasificación de algún modelo de ML; y así, a futuro poder predecir las calorías que presente algún plato en específico. Para esto se empezará describiendo los fundamentos teóricos del ML y su implementación en el procesamiento de imágenes de alimentos. Asimismo, se detallará el proceso de adquisición y limpieza da datos. Luego, se continuará con la descripción de imágenes asociadas a la comida, incluyendo la información necesaria que debe tener cada imagen para su etiquetamiento correspondiente.

Para el proceso de adquisición se realizarán consultas a bases de datos existentes (por ejemplo, repositorios de Kaggle o de universidades) que pueden ser de dos tipos de comida (por ejemplo, comida sin procesar y procesada). Para el proceso de limpieza de datos se considerará entre otros aspectos: i) la eliminación de imágenes que no cumplan con los criterios de calidad o se encuentren en un dominio diferente al de la comida que es objeto del

estudio, ii) imágenes duplicadas. En el procesamiento del conjunto de datos de imágenes de comida también se considerará añadir la descripción de cada imagen y el etiquetamiento correspondiente, a fin de que los datos puedan ser utilizados a futuro en algoritmos de aprendizaje supervisado o no supervisado según corresponda. Finalmente se espera obtener y exportar el conjunto de datos etiquetado y depurado en un formato adecuado para su uso en modelos de ML.

1.4 MARCO TEÓRICO

1.4.1 IMPORTANCIA DEL CONJUNTO DE DATOS EN EL MACHINE LEARNING

El aprendizaje automático, en líneas generales, consiste en el uso de métodos computacionales que aprovechan la experiencia previa para mejorar el desempeño o lograr predicciones precisas. En este contexto, la experiencia se refiere a la información previa disponible para el sistema de aprendizaje, que generalmente se presenta en forma de datos electrónicos recopilados y listos para su análisis. Estos datos pueden ser conjuntos de entrenamiento digitalizados que han sido etiquetados por personas, o cualquier otro tipo de información adquirida a través de la interacción con el entorno. En cualquier caso, la calidad y cantidad de estos datos son factores esenciales para el éxito de las predicciones realizadas por el sistema de aprendizaje [1].

Los conjuntos de datos que se hallan en entornos del mundo real suelen estar llenos de imperfecciones y enfrentan diversos problemas vinculados a la calidad de la información. Por ello, resulta crucial intentar garantizar que los datos mantengan altos estándares de calidad. Esto posibilitará que el sistema o algoritmo de ML, utilicen conjuntos de datos con precisión, posibilitando así la representación fiel y la realización de predicciones exactas sobre el fenómeno que se busca analizar [3].

Los datos en un proceso de ML son de mucha importancia ya que ayuda para:

1. **Entrenamiento del modelo:** La capacidad de un modelo de aprendizaje automático para ser preciso pude mejorar de manera significativa cuando se entrena con grandes volúmenes de datos. Si no se dispone de suficientes datos, se corre el riesgo de tomar decisiones basadas en pequeñas porciones de información, lo que podría llevar a interpretaciones erróneas de tendencias [4].
2. **Validación y evaluación:** Los datos juegan un papel crucial en la validación y la evaluación de la calidad y el desempeño de un modelo. Se recurre a conjuntos de

datos de prueba o de validación con el fin de determinar qué tan eficaz y preciso es el modelo en datos que no formaron parte del proceso de entrenamiento. Esta práctica es esencial para asegurarse de que el modelo sea verdaderamente efectivo y preciso [4].

3. **Aprendizaje supervisado y no supervisado:** En el aprendizaje supervisado, se entrega un conjunto de datos que incluye ejemplos con respuestas ya conocidas y etiquetadas. Por otro lado, en el aprendizaje no supervisado, se emplean datos que no cuentan con etiquetas para identificar patrones ocultos. En ambas modalidades, los datos son el componente fundamental necesario para llevar a cabo el proceso de aprendizaje [4].
4. **Mejora continua:** Los modelos de aprendizaje automático pueden perfeccionarse a lo largo del tiempo a medida que se les suministra una mayor cantidad de datos. Cuanta más información se recoja y se utilice para su formación y ajuste, mayor será la habilidad del modelo para tomar decisiones exactas.

1.4.2 PROCESO PARA LA OBTENCIÓN DE UNA BASE DE DATOS

Para el proceso de construcción de una base de datos correspondiente a imágenes se usa una metodología compuesta por etapas, como se muestra en resumen en la Figura 1.1.

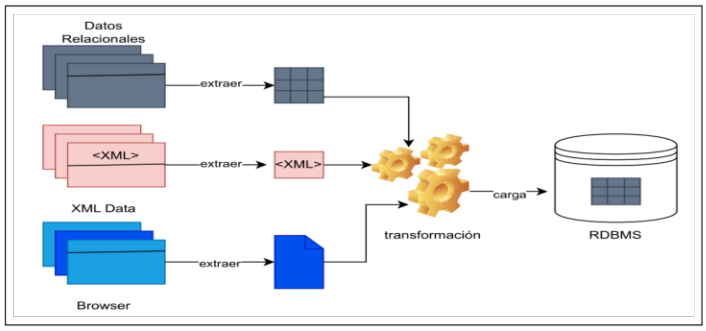

Figura 1.1: Un proceso de Extract, Transform and Load (ETL: Extracción, Transformación y Carga) puede recopilar información de diversas fuentes y modificarla antes de insertarla en un sistema de destino único como en un Relational Database Management System (RDBMS: Sistema de Gestión de Bases de Datos Relacionales), basado en [5].

1.4.2.1 Adquisición de los datos

En la etapa de extracción o adquisición de datos, se involucra en la recolección de datos provenientes de diversas fuentes. Estas fuentes pueden ser internas, tales como archivos almacenados en la nube. No obstante, también es común encontrar valiosas fuentes de datos externas, como servicios web proporcionados por colaboradores, proveedores o clientes, archivos de bases de datos de acceso público e incluso datos extraídos de la web,

como información de redes sociales y otros sitios web [6].

En el caso de una base de datos de imágenes, el proceso de extracción sigue un procedimiento similar a la de cualquier otra base que almacena números, texto, datos booleanos, etc. Para llevar a cabo esta tarea, es necesario definir previamente el propósito de la base de datos y explorar diversos repositorios en caso de que se busque utilizar una base de datos existente. Sin embargo, si se pretende crear la base de datos desde cero, el primer paso es la definición de las categorías que se emplearán. Estas categorías se convertirán en las etiquetas futuras de las imágenes. A partir de esta definición, se procede a buscar en varios motores y descargar las imágenes de acuerdo a la utilidad y el propósito específico de la base de datos. Por ejemplo, si se busca crear una base de datos de alimentos a partir de bases de datos existentes, se debe explorar los diversos repositorios, analizar cuáles cumplen con el objetivo y proceder a descargarlas. Algunas de las bases de datos de comida existentes se presenta en la Tabla 1.1.

Tabla 1.1: Información de algunos dataset de imágenes de comida disponibles en diferentes repositorios [7].

Dataset	Descripción	Número de imágenes
Generic (Fruits 360 Dataset)	Se trata de una extensa colección de información que incluye vegetales y frutas que desempeñan un papel fundamental en la rutina diaria de la gente. Esta base de datos organiza estos alimentos en categorías basadas en sus propiedades nutricionales, y cada imagen en ella muestra al menos una parte comestible de estos productos, destacando sus usos culinarios similares [8].	71 125
FLD-469	Es un grupo de información que utiliza múltiples conjuntos de metadatos, incluyendo lifelog, Wikipedia, USDA2 y Food-101. Este conjunto de datos se centra en la dieta y se utiliza para analizar con mayor precisión la recuperación intermodal en un entorno real de registro de alimentos a nivel global [9].	209 700
FoodX-251	Se trata de un conjunto de datos que consta de 251 categorías de alimentos específicos (preparados), con un total de 158,000 imágenes recopiladas de internet. Las clases son minuciosamente definidas y presentan similitudes visuales, como por ejemplo, diversas variedades de pasteles, sándwiches, pudines, sopas y pastas [10].	158 000
UNIMB 2016	Se trata de un conjunto de datos diseñado para evaluar algoritmos de reconocimiento de alimentos, los cuales son aplicables en contextos relacionados con el seguimiento de la dieta. Se escogió este dataset ya que cada imagen en el conjunto representa una bandeja que contiene una variedad de platos y alimentos dispuestos en diversas configuraciones y asemeja a los platos como lo encontraríamos en la cotidianidad [2].	1 027

1.4.2.2 Limpieza de datos

Este proceso busca mejorar la calidad de las bases de datos. Algunos de los aspectos de calidad que se buscan en este proceso incluyen mantener la información confidencial, asegurarse de que los datos estén escritos en el idioma adecuado para cada base de datos, mantener un nivel de información incompleta en cada registro por debajo del 10 % del total, evitar registros vacíos sin información, eliminar datos inusuales en bases de datos que no están diseñadas para manejarlos como datos no estructurados, datos de gran tamaño, datos desordenados, entre otros, y estandarizar los términos utilizados en las bases de datos [11].

Dentro del marco de una base de datos de imágenes y teniendo en cuenta la diversidad de métodos para crear o extraer estas imágenes, resulta esencial implementar controles específicos para asegurar la consecución de los objetivos predefinidos. A pesar de las numerosas consideraciones que se puedan tener en cuenta al extraer la información de manera óptima, ciertos factores como el formato de color, la presencia de imágenes duplicadas o el tamaño de las imágenes pueden tener un impacto significativo en la calidad del conjunto de datos [12]. En este proceso, se pueden emplear diversas herramientas según el objetivo deseado. Por ejemplo, si se pretende reducir el tamaño o modificar el formato de color de las imágenes, Python se presenta como una opción adecuada. En este caso, simplemente se cargaría el conjunto de datos y se codificaría la transformación deseada. Por otro lado, si la meta es eliminar imágenes duplicadas, se requeriría extraer las características de las imágenes utilizando modelos previamente entrenados como ResNet18 [13], y luego agrupar las imágenes mediante un algoritmo de vecinos cercanos.[1]

1.4.2.3 Procesamiento de los datos

Una vez que se han obtenido los datos finales seleccionados para su inclusión en la base de datos definitiva, es fundamental aplicar un formato específico, enriquecer los datos y prepararlos adecuadamente. Esta preparación es esencial para asegurar que los datos

[1] **Algoritmo de vecinos cercanos:** Algoritmo que emplea en clasificación y regresión[13]. En el caso de la clasificación, el punto se etiqueta con la clase que es más frecuente entre sus vecinos más próximos. Por otro lado, en la regresión, la predicción del valor del nuevo punto se obtiene calculando el promedio de los valores de sus vecinos más cercanos[14].

estén listos y optimizados para su análisis y consulta posterior, una vez que la base de datos final esté completamente configurada y operativa. Este proceso de acondicionamiento de los datos es crucial para garantizar su utilidad y efectividad en futuras investigaciones o aplicaciones.

En lo que respecta a bases de datos de imágenes, en esta etapa se realiza todo el procesamiento de las imágenes, lo cual engloba, entre otras tareas, la segmentación, el etiquetado y el tratamiento de los colores en las imágenes. Estos aspectos se abordarán con mayor detalle en la Sección 1.4.3.

En resumen, en esta etapa se lleva a cabo la preparación de los datos, con el propósito de facilitar su posterior inserción en la estructura de la base de datos final, que puede adoptar diversas formas, siendo una de las más comunes un archivo en formato .csv, entre otras opciones.

1.4.2.4 Carga de la data - construcción de la base de datos

La etapa de transferencia de datos involucra el traslado de la información procesada hacia su destino correspondiente. De manera efectiva, esta fase final recopila los datos que han sido extraídos, procesados y clasificados, para luego llevar a cabo la acción de entrega en el sistema o archivo de destino [15].

Por otra parte, es necesario generar fichas informativas (hoja de datos) que acompañen a cada conjunto de datos. Los responsables de crear estos conjuntos deben proporcionar detalles acerca de los procedimientos involucrados en su creación, distribución y mantenimiento, así como los posibles riesgos o perjuicios que podrían surgir al utilizar los datos. Esto permitirá que los usuarios puedan tomar decisiones fundamentadas acerca del empleo de un conjunto de datos. Estas fichas informativas servirán para mejorar la comunicación entre los creadores y los usuarios de los conjuntos de datos [15].

1.4.3 PROCESAMIENTO DE IMÁGENES PARA UNA BASE DE DATOS DE IMÁGENES

Cuando se describe el procesamiento de imágenes, es común idealizar los procedimientos en los que solo se manejan imágenes como entradas y salidas. Sin embargo, las computadoras, es decir, las máquinas, tienen la capacidad de trabajar con datos que provienen de diferentes formas y que generalmente no asociamos con imágenes, como ondas gam-

ma o señales de radio, por mencionar algunas. Esto abre la puerta a una amplia gama de aplicaciones diversas, como ultrasonidos, microscopía e incluso la creación de imágenes mediante software, entre otras posibilidades [12].

El interés en los métodos de procesamiento de imágenes digitales se origina a partir de dos áreas principales de aplicación. En primer lugar, se busca mejorar la calidad de la información visual para facilitar su comprensión por parte de los seres humanos. Además, se emplean estos métodos para el procesamiento de datos visuales con el fin de llevar a cabo tareas como almacenamiento, transmisión y extracción de información contenida en imágenes [16]. Los procesos manejados en el procesamiento digital están resumidos en: procesos de bajo, medio y alto nivel.

- ❒ **Proceso de bajo nivel:** La característica fundamental de este procedimiento en el contexto del procesamiento digital de imágenes es que tanto las entradas como las salidas consisten en imágenes. Está relacionado con las etapas de pre-procesamiento de imágenes, que incluyen operaciones elementales como la reducción de ruido[2], la mejora del contraste y la nitidez de una imagen [12].

- ❒ **Proceso de nivel medio:** Este nivel se destaca por usualmente utilizar imágenes como entrada, pero produce resultados que consisten en atributos derivados de esas imágenes, como por ejemplo, bordes y contornos. Entre las tareas que se realizan en este nivel se encuentran la segmentación, la descripción de objetos de manera apropiada para su procesamiento informático y la clasificación (reconocimiento) de objetos individuales.

- ❒ **Proceso de nivel alto:** Este nivel involucra la interpretación y dotación de significado a un grupo de objetos previamente identificados. Desde el análisis de imágenes hasta las capacidades cognitivas típicamente relacionadas con la visión humana son las actividades que se asocian con esta clasificación [16].

[2] **Reducción de ruido:** Es la acción de eliminar o reducir la presencia de distorsiones indeseadas o aleatorias que podrían afectar la calidad visual de la imagen [12].

1.4.3.1 Representación de imágenes digitales

Los valores de una imagen digital en un sistema de coordenadas (x, y) provienen de la función $f(x, y)$. En el contexto de una imagen, los valores máximos para x y y son M (filas) y N (columnas). Por lo tanto, se emplean números enteros para estas coordenadas discretas (x, y), con x variando de $\{0, 1, 2, ..., M-1\}$ y y de $\{0, 1, 2, ..., N-1\}$. El área en el plano real definida por las coordenadas de la imagen se conoce como el dominio espacial, y (x, y) son conocidas como las variables espaciales o coordenadas espaciales.

Una imagen digital puede ser conceptualizada como una función bidimensional $f(x, y)$ con coordenadas espaciales planas (x, y), y la magnitud de f se refiere como la intensidad o el nivel de gris de la imagen. La digitalización de una imagen implica que los valores de x, y, y la intensidad en f son cantidades finitas y discretas. En el contexto de una imagen digital, esta se desglosa en un número limitado de elementos (comúnmente píxeles), cada uno de los cuales posee una ubicación y un valor específico como se muestra en la Figura 1.2 [16].

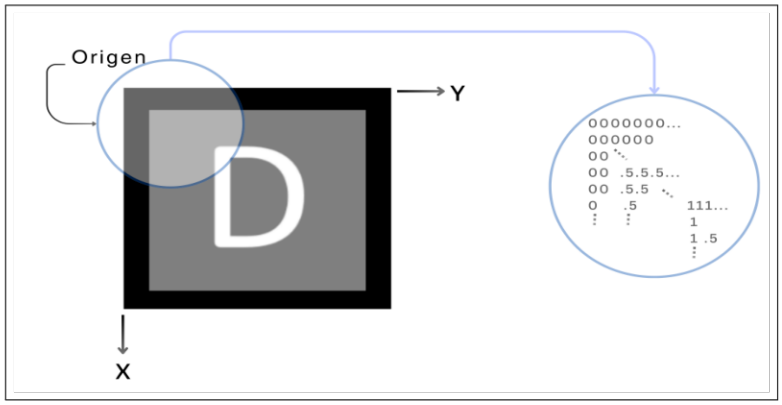

Figura 1.2: Imagen mostrada como una matriz numérica 2-D. (Los números 0, 0,5 y 1 representan negro, gris y blanco, respectivamente), basado en [16].

Otro de los aspectos a tener en cuenta en la representación de imágenes digitales es:

- **Color en el procesamiento de imágenes:** Hay dos dominios fundamentales que describen la manipulación de colores: i) el procesamiento de colores pseudos[3] y ii) el procesamiento a todo color. En el primer caso, se trata de asignar un color a una escala de grises específica o a un intervalo de intensidades. La escala de grises (o

[3] **Pseudo Color:** Proceso que asigna colores a la imagen según su nivel de gris, abarcando todo el espectro de colores RGB. Esto permite resaltar aspectos de la imagen que no serían claramente perceptibles en su color original [12].

intensidad) se refiere a una medida que varía desde el negro hasta el gris y finalmente el blanco. El segundo ámbito se caracteriza por la captura de imágenes a través de sensores, como una cámara digital o un escáner a color [12].

La capacidad de percepción visual en el ser humano se puede clasificar en tres categorías principales de colores: i) rojo, ii) verde y iii) azul. Todos los demás colores son percibidos por el ojo humano como una mezcla variable de estos colores primarios, debido a las propiedades de absorción [16].

1.4.3.2 Segmentación de imágenes

La segmentación automática es una de las tareas más desafiantes en el ámbito del procesamiento de imágenes digitales. Su propósito es dividir una imagen en regiones o componentes que la conforman. La mayoría de los algoritmos de segmentación se fundamentan en una de las dos propiedades fundamentales de los valores de intensidad en la imagen: i) la discontinuidad y ii) la semejanza [17].

La segmentación de una imagen implica dividirla en sus componentes individuales con el propósito de separar las áreas de interés del resto de la imagen. Este proceso puede implicar la detección de bordes, la segmentación en regiones, líneas, curvas, etc. Otra manera de entender la segmentación es como la asignación de clases a los diferentes píxeles de la imagen. Los aspectos fundamentales de la segmentación de imágenes pueden incluir la luminancia en imágenes monocromáticas, las componentes de color en imágenes a color, la textura, la forma, entre otros [17]. Como se mencionó previamente, los algoritmos utilizados para segmentar imágenes en escala de grises generalmente se basan en dos propiedades principales de los valores de intensidad: i) la discontinuidad y ii) la semejanza. En relación con la discontinuidad, este enfoque implica la subdivisión de una imagen en función de cambios abruptos en los niveles de gris. Los aspectos más notables relacionados con la discontinuidad incluyen la detección de puntos aislados, la identificación de líneas y la delimitación de bordes en la imagen. Por otro lado, en cuanto a la semejanza, se hace énfasis en la coherencia en los valores de intensidad de gris, y los principales métodos se centran en la umbralización[4], el crecimiento de regiones y la división, así como la fusión de regiones.

[4] **Umbralización:** Se destaca como uno de los procedimientos fundamentales en la segmentación de imágenes. El umbral se define como una función que transforma una imagen con diversas tonalidades en una representación en blanco y negro [17].

Por otro lado, a lo largo de las últimas décadas, se han desarrollado diversas técnicas de segmentación que se pueden clasificar en tres categorías: i) aquellas orientadas a píxeles, ii) a bordes y iii) a regiones. Entre las diversas técnicas destacadas se encuentra el método de la línea divisoria de aguas (watershed). Este método, partiendo de los mínimos[5]. presentes en la imagen, simula el flujo gradual de agua en un valle hasta alcanzar valles adyacentes. Otro enfoque es la detección de bordes de regiones, que se logra buscando máximos en el gradiente de la imagen o cruces por cero en la segunda derivada de la misma. También se emplean filtros que optimizan una función de costo, considerando tanto la precisión en la localización del borde como la cantidad de bordes detectados. Además, se utiliza la detección de regiones mediante la agrupación de píxeles vecinos que comparten características similares, conocida como Región Growing[6] [17].

Los modelos de borde que se usan en un proceso de segmentación se agrupan en función de sus patrones de intensidad. Un borde de transición se identifica por un cambio de nivel de intensidad que idealmente sucede en la distancia de un solo píxel. En la Figura 1.3 se muestra tres ejemplos claros sobre los cambios de nivel de intensidad, la imagen a) ilustra una sección de un escalón vertical en el borde, junto con un perfil de intensidad horizontal que atraviesa dicho borde. Los bordes de escalones, como los que se observan en imágenes generadas por computadora para aplicaciones como el modelado de sólidos y la animación, se caracterizan por su nitidez y perfección, y se espera que aparezcan en una distancia de un solo píxel. La imagen b) corresponde a un borde que tienen un perfil de intensidad en forma de rampa que representan imágenes que tienen bordes borrosos y con ruido, situaciones que se determinan por limitaciones en el mecanismos del enfoque como por ejemplo los lentes en el caso de imágenes ópticas. Otro tipo de borde es conocido como borde del techo, el cual exhibe las características representadas en la imagen c). Los bordes del techo se describen como patrones lineales que atraviesan una zona, y la anchura del borde está definida por la espesura y definición de la línea. Los bordes del tejado son evidentes en situaciones como imágenes de alcance, donde objetos alargados y delgados, como tuberías, se encuentran más próximos al sensor que el fondo, es decir la relación espacial entre los objetos y el dispositivo de captura de la imagen (el sensor), como

[5] **Mínimo y/o máximo de una imagen:** Son los extremos de intensidad de los píxeles. El mínimo representa la menor intensidad, usualmente asociada con áreas oscuras, mientras que el máximo corresponde a la mayor intensidad, típicamente relacionada con áreas más brillantes [17].

[6] **Región Growing:** Es un método que implica la agrupación de píxeles contiguos que comparten características similares con el fin de crear áreas más extensas y relevantes en la imagen [17].

las paredes. Debido a su mayor brillo, las tuberías generan una imagen que se asemeja al modelo [16].

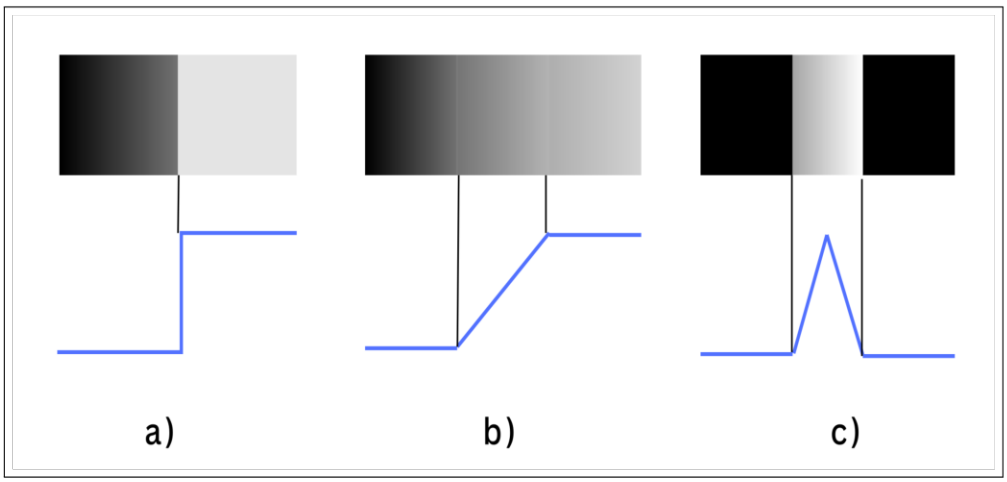

Figura 1.3: De izquierda a derecha, modelos (representaciones ideales) de un escalón a), una rampa b) y un borde de techo c), y sus perfiles de intensidad correspondientes, basado en [16].

Otras características o aspecto a tomar en cuenta en la segmentación de imágenes son:

- **Componentes conexos o conectados:** Dos píxeles se consideran conectados cuando, al ser evaluados, se encuentran adyacentes en algún sentido, teniendo en cuenta su vecindad y sus niveles de intensidad. Esta intensidad puede referirse tanto al nivel de gris como a la representación binaria en blanco y negro. La similitud entre sus niveles de intensidad se determina mediante una regla específica, que generalmente se basa en la igualdad. Si p y q son píxeles pertenecientes a un subconjunto S de una imagen, se establece que p está conectado con q dentro de S si existe un camino (compuesto únicamente por píxeles de S) que los vincula. Se denomina componente conexo al conjunto de píxeles S conectados a p (cualquier píxel p perteneciente a S) [16].

- **Punto, línea y detección de bordes:** Los puntos aislados, las líneas y los bordes representan elementos de interés en una imagen. Un punto aislado puede visualizarse como un solo píxel rodeado por píxeles del fondo o primer plano. De manera similar, una línea generalmente se compone de un segmento delgado de borde, y la intensidad del fondo en ambos lados de la línea es distinta de la intensidad de los píxeles que componen la línea. Los detectores de bordes son herramientas de procesamiento de imágenes que se diseñan para identificar conjuntos de píxeles de borde conectados, donde se produce un cambio abrupto en la intensidad en contraste con los píxeles del fondo en la imagen [12].

Cuando se trabaja con imágenes y se utiliza el color como base, se pretende estimar el color "medio"que se desea identificar y dividir en segmentos. El proceso se centra en un rango de colores particulares en una imagen a color, representado por un vector RGB. La meta de la segmentación es asignar a cada píxel de la imagen una categoría específica, lo cual se logra al evaluar la similitud entre ellos.

Sin bien todas estas técnicas de segmentación dan excelentes resultados hoy en día con la llegada de las inteligencias artificiales (IA: Artificial Intelligence) este proceso a tenido mejoras radicalmente. Insertar algoritmos como los de aprendizaje profundo permite una interpretación más avanzada y automatizada de las características visuales en las imágenes [12].

Con la inteligencia artificial el problema de variaciones en la iluminación, ruido en las imágenes o la presencia de objetos superpuestos desaparecen ya que los modelos de segmentación pueden aprender patrones complejos y adaptarse a diversas condiciones. Además, los algoritmos basados en inteligencia artificial pueden realizar tareas de segmentación de manera más rápida y precisa, lo que mejora la eficiencia operativa en una variedad de aplicaciones.

Para la parte de segmentación de este trabajo de integración curricular, se usa Modelo de Segmentación de Cualquier Cosa (SAM: Segment Anything Model), del cual se proporcionará una descripción más detallada en la Sección 1.6. SAM es una eficaz herramienta para la segmentación de imágenes, empleando un codificador de imágenes robusto para crear una representación incrustada de la imagen original. Esta representación es fácilmente accesible mediante diversos tipos de entradas, lo que facilita la generación de máscaras[7] de objetos en tiempo real. La elección de este modelo para el proyecto se basó en su capacidad para generar máscaras rápidamente, evitando así el uso de modelos tradicionales que a menudo requieren intervención manual.

Este modelo ha alcanzado un hito al crear el conjunto de datos de segmentación más extenso hasta la fecha, con más de mil millones de máscaras aplicadas a 11 millones de imágenes, todas con licencia y respetando la privacidad de los usuarios [18]. Esto destaca

[7] **Máscara:** Es una representación gráfica de la disposición espacial de un objeto en una imagen. La máscara consiste esencialmente en una imagen binaria en la que los píxeles que forman parte del objeto se identifican de manera específica (por ejemplo, con el valor 1), mientras que los píxeles que no forman parte del objeto tienen otro valor (por ejemplo, 0) [18].

cómo la IA está transformando positivamente el proceso de segmentación de imágenes, logrando hacerlo más rápido y preciso.

1.5 HERRAMIENTAS DE PROGRAMACIÓN

1.5.1 PYTHON

Python es un versátil lenguaje de programación de alto nivel que se utiliza ampliamente en el desarrollo de software, gracias a su productividad, potencia y flexibilidad. Su sintaxis es clara y concisa, y una de sus ventajas más notables es que no necesita ser compilado, ya que es un lenguaje de programación interpretado[8] [19]. Python se convierte en una elección muy popular en diversas áreas, como el desarrollo web, la automatización de scripts y procesos, especialmente en ML, gracias a su amplia gama de bibliotecas y macros que simplifican la codificación y aceleran el proceso de desarrollo [20].

1.5.1.1 Numpy

NumPy, que es una abreviatura de Numerical Python, desempeña un papel fundamental en la computación científica en Python, esta librería ofrece diferentes funcionalidades como:

- ❒ Una estructura de datos de matriz multidimensional eficiente y veloz denominada ndarray.
- ❒ Capacidad para realizar cálculos elemento a elemento y operaciones matemáticas entre matrices.
- ❒ Herramientas para leer y escribir conjuntos de datos basados en matrices en dispositivos de almacenamiento.
- ❒ Funciones para llevar a cabo operaciones de álgebra lineal.
- ❒ Transformada de Fourier y generación de números aleatorios.
- ❒ Facilita la integración de código en lenguajes como C, C++ y Fortran en Python [21].

[8] **Lenguaje de programación interpretado:** Es aquel cuyo código fuente no se compila completamente en un archivo ejecutable antes de su ejecución [19].

1.5.1.2 Torch

PyTorch es una amplia biblioteca de código abierto en el ámbito del aprendizaje profundo y la IA, ofreciendo recursos y estructuras que simplifican la elaboración y entrenamiento de redes neuronales y modelos de aprendizaje profundo. Inicialmente PyTorch está orientado hacia flujos de trabajo de investigación, PyTorch ha sido mejorado con un tiempo de ejecución en C++ de alto rendimiento, lo que permite implementar modelos para inferencia sin depender de Python, así como diseñar y entrenar modelos en C++ [22].

1.5.1.3 matplotlib.pyplot

En Python, existe un módulo dentro de la biblioteca Matplotlib que ofrece una interfaz basada en objetos para la creación de gráficos y representaciones visuales de datos de manera eficiente y sencilla. Matplotlib destaca como una de las herramientas más populares en Python para la elaboración de gráficos y visualizaciones [22].

1.5.1.4 OpenCv

OpenCV es una herramienta esencial en el ámbito de la visión por computadora, ya que se trata de una biblioteca de código abierto de gran relevancia. Se emplea extensamente en el procesamiento de imágenes y la visión por computadora, ofreciendo un marco de trabajo que permite manipular imágenes y videos de manera flexible. Puede aplicar tanto los algoritmos preexistentes en OpenCV como desarrollar los suyos propios, sin la necesidad de gestionar la asignación y liberación de memoria para sus datos visuales [23].

1.5.1.5 h5py

Es una herramienta de programación en Python diseñada para manipular archivos de datos que siguen el formato Formato de Datos Jerárquico versión 5 (HDF5: Hierarchical Data Format versión 5). Esta librería ofrece a los usuarios de Python una forma práctica de crear, leer y escribir archivos en formato HDF5, además de permitir la interacción con los conjuntos de datos almacenados en estos archivos. Con H5py, es posible acceder a los grupos y conjuntos de datos dentro de un archivo HDF5[9], así como realizar diversas operaciones de lectura y escritura en ellos.

[9] **HDF5:** Es un formato de archivo diseñado para almacenar y organizar grandes cantidades de datos de manera eficiente y flexible. Está especialmente adaptado para el almacenamiento de conjuntos de datos complejos y estructurados [23].

1.6 SEGMENTAR CUALQUIER MODELO (SAM)

Modelo de Segmentación de Cualquier Cosa (SAM) es un proyecto que introduce una nueva tarea, modelo y conjunto de datos para la segmentación de imágenes. Con un enfoque en la eficiencia y la recopilación de datos, SAM ha logrado crear el conjunto de datos de segmentación más extenso hasta la fecha, destacándose con más de mil millones de máscaras en 11 millones de imágenes, todas con licencia y respetando la privacidad de los usuarios [18].

El modelo de SAM ha sido diseñado y entrenado con la premisa de ser rápido y versátil, lo que le permite adaptarse sin problemas a nuevas distribuciones de imágenes y tareas. Las capacidades de SAM han sido evaluadas en diversas tareas, revelando un rendimiento excepcional, a menudo comparable e incluso superior a los resultados obtenidos en enfoques completamente supervisados en el campo de la segmentación de imágenes [18].

La descripción general SAM revela un enfoque altamente eficiente en la segmentación de imágenes debido a su capacidad para generar máscaras con gran velocidad y precisión. Esto evita la necesidad de llevar a cabo procesos convencionales que consumirían mucho más tiempo. Esta representación se puede consultar de manera ágil utilizando una variedad de mensajes de entrada , lo que permite la generación de máscaras de objetos a una velocidad amortizada en tiempo real, es decir, aunque en algunas ocasiones pueda tomar más tiempo generar una máscara, en términos generales se logra hacer de manera rápida, lo que posibilita su utilización en aplicaciones que demandan respuestas velozmente [18].

Uno de los puntos destacados de SAM es su capacidad para lidiar con imágenes ambiguas que corresponden a la presencia de más de un objeto en una imagen. En tales casos, SAM puede generar múltiples máscaras válidas y proporcionar puntuaciones de confianza asociadas, lo que lo convierte en una herramienta versátil y precisa para abordar problemas de segmentación de imágenes en diversas aplicaciones.

SAM tiene tres componentes mostrados en la Figura 1.4. Un codificador de imágenes (image encoder), un codificador de avisos flexible (flexible prompt encoder) y un decodificador de máscara rápido (fast mask decoder).

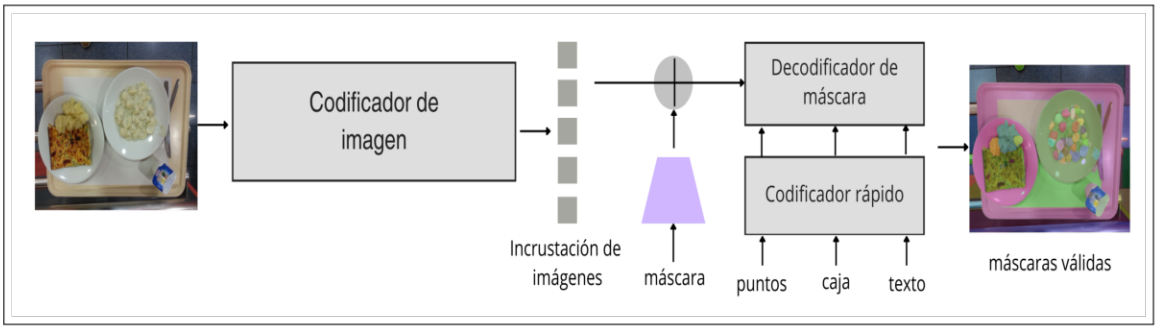

Figura 1.4: Descripción general del modelo SAM. Un codificador de imágenes pesado genera una imagen incrustada que luego puede consultarse de manera eficiente mediante una variedad de mensajes de entrada para producir máscaras de objetos a una velocidad amortizada en tiempo real. Para mensajes ambiguos correspondientes a más de un objeto, SAM puede generar múltiples máscaras válidas y puntuaciones de confianza asociadas, basado en [18].

- **Codificador de imágenes:** Impulsados por el interés en lograr una mayor escalabilidad y aprovechar los eficaces métodos de preentrenamiento como las Redes Neuronales Convolucionales Preentrenadas o como Autoencoders Preentrenados, se ha empleado un Transformador de Visión (ViT: Vision Transformer) preentrenado, específicamente el ViT MAE [24], [25], con ajustes mínimos para procesar entradas de alta resolución. El codificador de imágenes se ejecuta una sola vez por imagen y está listo para su aplicación antes de requerir la intervención del modelo. Este enfoque optimizado tiene como objetivo maximizar la eficiencia y el rendimiento del sistema, permitiendo un procesamiento eficaz de imágenes de alta calidad.

- **Decodificador de máscaras:** El bloque decodificador modificado incorpora dos tipos de atención: i) auto atención rápida y ii) atención cruzada en dos direcciones (incrustación de imagen y viceversa) para actualizar todas las incrustaciones relevantes. Después de ejecutar dos bloques de este proceso, se aumenta la muestra de la imagen incrustada, y un Perceptrón Multicapa (MLP: Multilayer Perceptron) asigna el token de salida a un clasificador lineal[10] dinámico. Este clasificador calcula posteriormente la probabilidad de primer plano de la máscara en cada ubicación de la imagen [18].

- **Flexible prompt encoder:** Existen dos conjuntos de indicaciones en consideración: uno de ellos, conocido como escaso (sparse), abarca elementos como puntos, cuadros y texto, mientras que el otro, llamado denso (dense), se refiere a las máscaras. En este contexto, se presenta cómo se representan los puntos y los cuadros median-

[10] **Clasificador lineal:** Es un tipo de modelo de aprendizaje automático que realiza una clasificación basada en una función lineal de las características de entrada [18].

te codificaciones posicionales, las cuales se combinan con incrustaciones específicas diseñadas para cada tipo de mensaje, obteniendo así las máscaras válidas[11] [18].

[11] **Máscara válida:** Es una matriz que permite seleccionar o filtrar regiones específicas de una imagen de acuerdo con los criterios establecidos por el usuario [18].

2 METODOLOGÍA

Este capítulo describe el proceso de construcción del conjunto de datos de imágenes de alimentos. En la Figura 2.1 se puede ver la metodología que se siguió para la obtención de la base de datos, dicha metodología se basó en el proceso Extracción, Transformación y Carga (ETL: Extract, Transform and Load).

El proceso de Extracción, Transformación y Carga (ETL) se emplea para integrar y preparar datos con el fin de análisis y gestión de datos. Este proceso se compone principalmente de tres fases fundamentales [26]:

1. **Extracción:** La información es obtenida de una variedad de fuentes, incluyendo bases de datos, archivos de sistemas, y aplicaciones en línea, entre otros recursos.

2. **Transformación:** Los datos son modificados con el fin de adecuarlos para su análisis o para su incorporación en un sistema específico. Este proceso implica acciones como la depuración de datos, la conversión entre formatos, la eliminación de duplicados y la estandarización de los esquemas de datos, entre otras operaciones.

3. **Carga:** La información es transferida al sistema final previsto, que puede ser un repositorio de datos, una plataforma de análisis, una base de datos operativa, u otras alternativas.

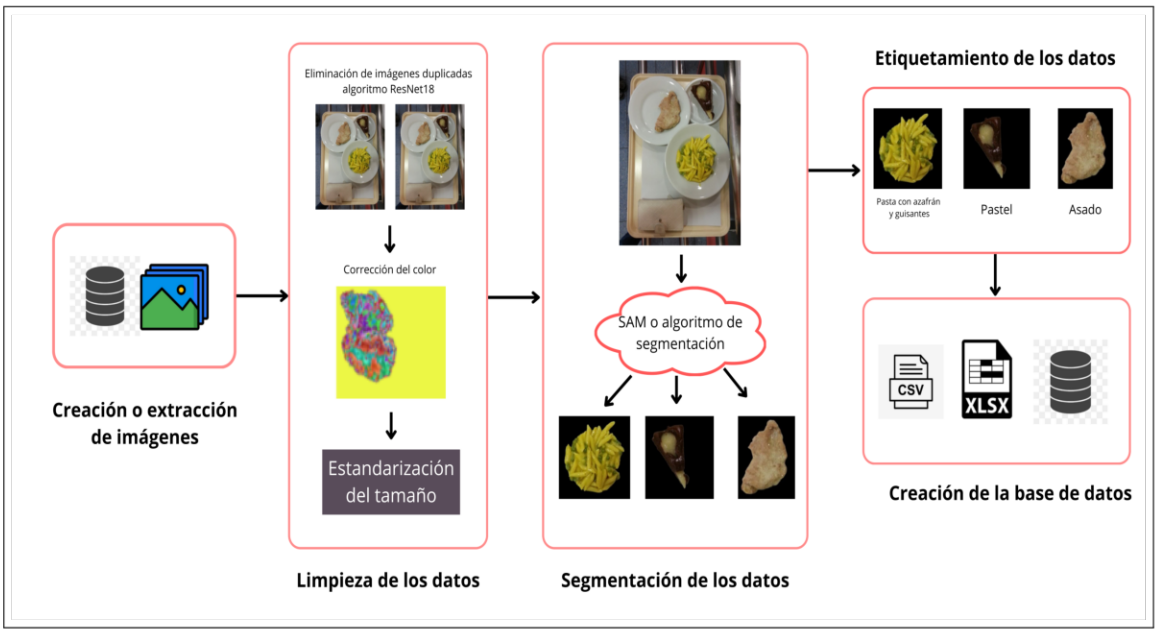

Figura 2.1: Metodología seguida para la creación de la base de datos.

La base de datos que se generará se derivará de una base de datos preexistente denominada UNIMIB 2016 una base de datos de comida italiana, la cual fue recopilada en un entorno correspondiente a un restaurante real. Este conjunto de datos destaca por presentar imágenes en las que diversos alimentos se disponen en bandejas, con algunos elementos como frutas, pan y postres ubicados directamente sobre las bandejas en lugar de platos convencionales [2].

Se emplea la totalidad de las imágenes proporcionadas por la base de datos, las cuales ascienden a 1,027. En este contexto, no se abordaron los procedimientos de recolección ni depuración de cada imagen para la construcción de la base de datos. En su lugar, se optó por tomar la totalidad de las imágenes y comenzar el proceso de segmentación que corresponde a la fase de procesamiento de datos. Sin embargo, cabe mencionar que para la creación de la base de datos UNIMIB 2016, se recopilaron un total de 1,442 imágenes. Estas imágenes fueron sometidas a una fase de control de calidad en la cual se eliminaron aquellas borrosas y las fotografías duplicadas. Posteriormente, tras completar esta fase, se obtuvo un conjunto de datos final conformado por 1,027 imágenes, distribuidas en 73 categorías de alimentos [2].

En la construcción de la base de datos actual, se utilizaron la mayoría de las etiquetas proporcionadas por la base de datos original, ajustándolas a la terminología en español y añadiendo algunas adicionales. Como resultado, la base de datos final quedó conformada por 77 etiquetas.

2.1 PROCESO GENERAL

La Figura 2.2 presenta el procedimiento para la construcción de una base de datos de imágenes. Se utilizó el Modelo y Notación de Procesos y Negocio (BPMN: Business Process Modeling Notation[1]).

[1] **BPMN:** Representación visual diseñada para uniformizar la conceptualización de los procedimientos empresariales mediante un idioma compartido, con el fin de simplificar la interpretación del rendimiento y la evolución de una o más tareas [27].

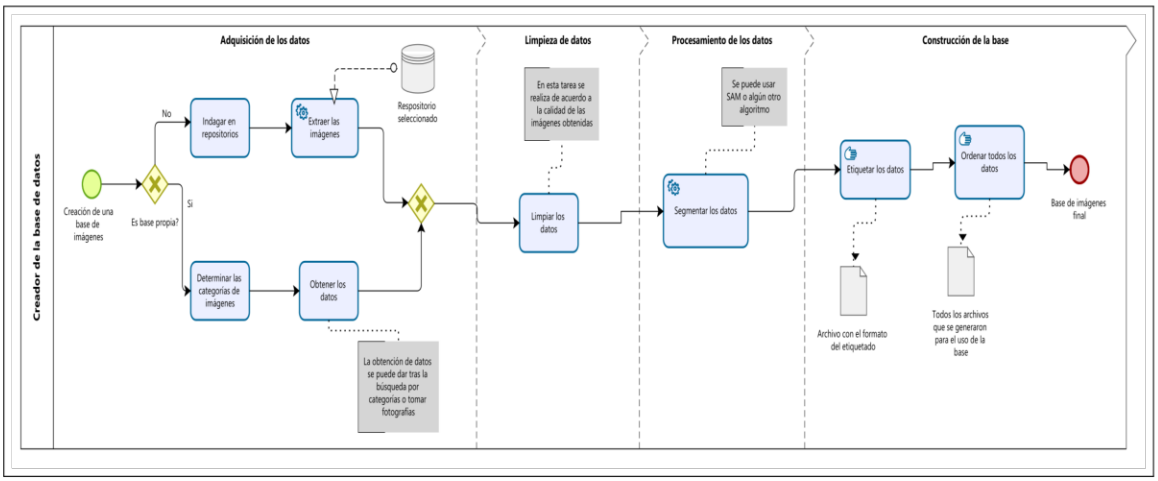

Figura 2.2: Proceso para la creación de la base de datos de imágenes.

En el proceso de la Figura 2.2 se observa que solo participa un rol que corresponde a la persona o entidad encargada de la creación de la base de datos de imágenes.

El proceso se divide en cuatro fases: adquisición de datos, limpieza de datos, procesamiento de datos y construcción de la base. Comienza con la fase de adquisición de datos, donde el creador de la base establece la fuente de obtención de los datos antes de crear la base propiamente dicha. Posteriormente, se avanza a la fase de limpieza de datos, donde se analiza y determina qué acciones se deben llevar a cabo según el resultado de la fase anterior. En la fase de procesamiento de datos, se realizan diversas acciones, incluida la segmentación de imágenes utilizando SAM u otros algoritmos pertinentes. Finalmente, en la fase de construcción de la base, se lleva a cabo el etiquetado y ordenamiento de los datos en los formatos de almacenamiento correspondientes.

2.2 MÁSCARAS DE OBJETOS CON SAM

Una vez que se han elegido las imágenes con las que se llevará a cabo el presente Trabajo de Integración Curricular, abarcando los pasos de adquisición y limpieza de datos, el proceso continúa con el procesamiento de la información. En esta etapa inicial, se procede con la segmentación de las imágenes.

Para el proceso de segmentación de las imágenes, se empleó SAM, una inteligencia artificial que posibilita la automatización de la detección de máscaras de los objetos presentes en una imagen. A continuación, se describe el proceso en código de SAM utilizado para la creación de dichas máscaras.

SAM genera predicciones de máscaras de objetos basadas en indicaciones que especifican el objeto deseado [18]. Inicialmente, el modelo transforma la imagen en una imagen

incrustada, lo que permite la creación eficiente de máscaras de alta calidad cuando se le solicita.

A continuación, se presenta un ejemplo de cómo SAM detecta máscaras en una imagen utilizando Google Colab. En primer lugar, es fundamental seleccionar una GPU en Colab para optimizar el hardware. Esta funcionalidad resulta especialmente beneficiosa al abordar tareas que requieren un gran poder de procesamiento, como el entrenamiento de modelos de aprendizaje profundo o la realización de operaciones intensivas en datos. Las GPUs han demostrado ser más eficaces en estas actividades en comparación con las tradicionales unidades de procesamiento central (CPU).

El fragmento de código que se muestra en el Código 2.1 se emplea en un entorno de Colab con el propósito de realizar la instalación de diversas bibliotecas, la descarga de una imagen de muestra, y la obtención de un modelo pre-entrenado.

1. En las línea 2 y 3 se cargan las bibliotecas PyTorch y Torchvision, comúnmente empleadas en el ámbito del aprendizaje profundo.
2. En las línea 4 y 5 se muestran las versiones instaladas de PyTorch y Torchvision.
3. En la línea 6 se comprueba la disponibilidad de la aceleración de hardware CUDA[2], que se utiliza para mejorar el rendimiento del aprendizaje profundo en unidades de procesamiento gráfico (GPU).
4. En la línea 7 se incluye el módulo sys para acceder a funciones relacionadas con el sistema.
5. En la línea 8 se emplea el comando pip para llevar a cabo la instalación de las bibliotecas OpenCV y Matplotlib, las cuales son ampliamente utilizadas en el procesamiento de imágenes y representación gráfica.
6. En la línea 9 se usa pip para llevar a cabo la instalación de un paquete a partir de un repositorio en GitHub proporcionado en forma de URL.
7. En la línea 11 se genera un directorio en el sistema denominado “images“ .

[2] **Aceleración de hardware CUDA:** Es el uso de la arquitectura CUDA desarrollada por NVIDIA, esta metodología posibilita llevar a cabo cálculos intensivos de forma más efectiva al hacer uso de la capacidad de procesamiento altamente paralelo de las unidades de procesamiento gráfico (GPU), lo que puede conducir a una ejecución más veloz de algoritmos y aplicaciones específicas en contraste con la ejecución en una unidad central de procesamiento (CPU) convencional [28].

8. En la línea 12 se descarga una imagen desde internet y almacénala en la carpeta ïmages".

9. En la línea 14 se descarga un archivo de la web con el nombre “ sam_vit_h_4b8939.pth“ el cual es el modelo previamente entrenado.

Existen tres variantes del modelo disponibles, cada una con diferentes dimensiones en su núcleo central estos son: Default o vit_h, vit_l y vit_b [18].

```
1  if using_colab:
2      import torch
3      import torchvision
4      print("PyTorch version:", torch.__version__)
5      print("Torchvision version:", torchvision.__version__)
6      print("CUDA is available:", torch.cuda.is_available())
7      import sys
8      !{sys.executable} -m pip install opencv-python matplotlib
9      !{sys.executable} -m pip install 'git+https://github.com/facebookresearch/
           segment-anything.git'
10
11     !mkdir images
12     !wget -P images https://raw.githubusercontent.com/facebookresearch/segment-
           anything/main/notebooks/images/dog.jpg
13
14     !wget https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth
```

Código 2.1: Preparación del entorno de Colab para la generación de máscaras de una imágen en particular.

El Código 2.2 importa bibliotecas especificas. La funcionalidad efectiva del programa dependerá de cómo se utilicen estas bibliotecas en el código. Estas bibliotecas son típicas en aplicaciones vinculadas a la visión por computadora, el aprendizaje profundo y el procesamiento de imágenes.

```
1  import numpy as np
2  import torch
3  import matplotlib.pyplot as plt
4  import cv2
```

Código 2.2: Importación de bibliotecas ideales para el proceso de obtención de máscaras de una imágen.

El fragmento del Código 2.3 indica la creación de una función llamada *show_anns* diseñada para mostrar las anotaciones (annotations) de objetos en una imagen.

- La función recibe un parámetro denominado *anns*, el cual debe ser una lista que contiene anotaciones de objetos. En primer lugar, comprueba si la lista de anotaciones (anns) está vacía. En caso de no contener anotaciones, la función se detiene de inmediato con la instrucción *return*, lo que implica que no llevará a cabo ninguna acción si no hay objetos para mostrar. Si existen elementos en la lista, entonces se procede a organizarlas en base al tamaño del objeto al que están asociadas. Estas anotaciones se disponen en un orden decreciente, lo que conlleva que el objeto de mayor tamaño será el primero en mostrarse en la visualización. Se adquiere una referencia al eje actual del gráfico en uso mediante el empleo de *plt.gca()* de Matplotlib. En la línea 6 se desactiva la función de ajuste automático del escalado en el eje actual mediante la instrucción *ax.set_autoscale_on(False)*. Esto impide que Matplotlib adapte automáticamente el tamaño del gráfico en relación a los datos, siendo especialmente útil cuando se pretende superponer objetos en una imagen sin modificar sus dimensiones.

- En el Código 2.4 se genera una matriz denominada *img*, la cual posee las dimensiones idénticas a las de la primera anotación en la lista *sorted_anns*. Esta matriz consta de cuatro canales, y el cuarto canal (índice 3) se emplea para denotar la transparencia (alfa). El ciclo for recorre las anotaciones presentes en la lista *sorted_anns*. Para cada anotación, se extrae la máscara de segmentación del objeto, la cual se encuentra almacenada en *ann['segmentation']*.

- Se crea un color al azar para simbolizar el objeto. Este proceso consiste en generar tres valores aleatorios comprendidos entre 0 y 1 para los canales de color rojo, verde y azul, y, a continuación, se asigna un valor constante de 0.35 al canal alfa, el cual regula la opacidad del objeto.

- La máscara de segmentación del objeto se emplea para modificar la matriz *img* con el color al azar adecuado, lo que resulta en la inclusión del objeto en la imagen. Por último, se emplea *ax.imshow(img)* para exhibir la imagen resultante en el gráfico actual, la cual presenta los objetos coloreados de acuerdo a sus anotaciones y dispuestos en un orden decreciente basado en su tamaño.

```
def show_anns(anns):
    if len(anns) == 0:
        return
    sorted_anns = sorted(anns, key=(lambda x: x['area']), reverse=True)
    ax = plt.gca()
    ax.set_autoscale_on(False)
```

Código 2.3: Creación de la función llamada show_anns diseñada para mostrar las anotaciones de objetos en una imagen.

```
    img = np.ones((sorted_anns[0]['segmentation'].shape[0], sorted_anns[0]['
        segmentation'].shape[1], 4))
    img[:,:,3] = 0
    for ann in sorted_anns:
        m = ann['segmentation']
        color_mask = np.concatenate([np.random.random(3), [0.35]])
        img[m] = color_mask
    ax.imshow(img)
```

Código 2.4: Continuación de la creación de la función llamada show_anns diseñada para mostrar las anotaciones de objetos en una imagen.

En el Código 2.5 se carga la imagen con la que se trabaja llamada en este caso 603.jpg (Figura 2.3), la transforma en el formato de color RGB, la presenta en una figura de Matplotlib con ejes deshabilitados y, por último, la exhibe en una ventana gráfica.

```
image = cv2.imread('content/drive/MyDrive/Tesis_Prueba/603.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(20,20))
plt.imshow(image)
plt.axis('off')
plt.show()
```

Código 2.5: Muestra la imagen a la que se le va a segmentar.

Figura 2.3: Imagen 603.jpg usada en el Código 2.5.

El Código 2.6 se lleva a cabo diversas operaciones vinculadas a la biblioteca segment_anything (modelo SAM), el cual engloba funciones y clases destinadas a la segmentación de objetos en imágenes.

```
1  import sys
2  sys.path.append("..")
3  from segment_anything import sam_model_registry, SamAutomaticMaskGenerator,
       SamPredictor
4
5  sam_checkpoint = "sam_vit_h_4b8939.pth"
6  model_type = "vit_h"
7
8  device = "cuda"
9
10 sam = sam_model_registry[model_type](checkpoint=sam_checkpoint)
11 sam.to(device=device)
12
13 mask_generator = SamAutomaticMaskGenerator(sam)
```

Código 2.6: Operaciones vinculadas a la biblioteca segment_anything (modelo SAM).

Se incorporan tres elementos del módulo segment_anything:

- ***sam_model_registry:*** Un conjunto de modelos de segmentación preestablecidos guardados en forma de un diccionario.
- ***SamAutomaticMaskGenerator:*** Una clase que produce máscaras automáticas para

objetos en imágenes mediante la aplicación de un modelo de segmentación.

- ***SamPredictor:*** Una clase diseñada para llevar a cabo predicciones de segmentación en imágenes.

En la línea 5 del Código 2.6 se crea una variable llamada *sam_checkpoint* que guarda el nombre de un archivo de control (checkpoint) del modelo de segmentación llamado *sam_vit_h_4b8939.pth*. Este archivo incluye los valores de los parámetros del modelo previamente entrenado. En la línea 6 se establece una variable llamada *model_type* que indica el tipo de modelo de segmentación que se empleará, y en este caso, se configura como *vit_h*.

En la línea 8 del Código 2.6 se establece una variable denominada *device* que determina el dispositivo de procesamiento a emplear para ejecutar el modelo de segmentación. En este caso, se configura como CUDA, indicando la posible utilización de una GPU si está disponible con el fin de acelerar el proceso. La línea 10 instancia el modelo de segmentación utilizando el tipo de modelo y el archivo de control mencionados. El modelo se carga desde el archivo de control y se guarda en la variable llamada sam.

El modelo SAM se traslada al dispositivo de procesamiento indicado en la variable *device.* Finalmente se instancia la clase *SamAutomaticMaskGenerator* utilizando el modelo de segmentación sam previamente creado. Esta instancia se guarda en la variable *mask_generator* y se empleará para producir máscaras automáticas de objetos en imágenes utilizando el modelo sam.

En el Código 2.7 en la línea 1 se invoca el método *generate* de la instancia de *SamAutomaticMaskGenerator*, pasando una imagen que está referenciada por la variable *image* (Figura 2.3) como argumento. Este método se emplea para crear máscaras automáticas que identifican los objetos en la imagen, y las máscaras se guardan en la variable *masks*. Luego se muestra en la salida estándar la extensión de la lista de máscaras (masks). Esto dará como resultado la cantidad de máscaras generadas, es decir, el número de objetos detectados en la imagen que en este caso son 99 máscaras.

Finalmente se imprimen las claves del primer diccionario[3] contenido en la lista de más-

[3] **Diccionario:** Es una estructura de datos que facilita el almacenamiento de información de manera vinculada, donde los datos se guardan en forma de pares clave-valor. Cada entrada en el diccionario se compone de una clave y un valor correspondiente. La clave funciona como un identificador único que se

caras (masks[0]). Cada máscara representa un diccionario que incluye detalles acerca de un objeto detectado en la imagen. Esto mostrará las claves o propiedades disponibles en el diccionario de la primera máscara, ofreciendo información sobre qué datos se han generado y están disponibles en relación con el objeto detectado.

```
masks = mask_generator.generate(image)

print(len(masks))
print(masks[0].keys())
```

Código 2.7: Generación de las máscaras que se detetacta en la figura utilizada.

El Código 2.8 presenta la imagen acompañada de las anotaciones (máscaras) de los objetos detectados. En primer lugar, se crea una figura de Matplotlib, se exhibe la imagen original en la figura, se superponen las anotaciones utilizando la función *show_anns*, se desactivan los ejes y se muestra la figura en una ventana gráfica (Figura 2.4).

```
plt.figure(figsize=(20,20))
plt.imshow(image)
show_anns(masks)
plt.axis('off')
plt.show()
```

Código 2.8: Imagen acompañada de las máscaras de los objetos detectados.

Figura 2.4: Imagen resultante de la segmentación de la Figura 2.3 que se muestra en el Código 2.8.

emplea para acceder al valor relacionado [19].

2.2.1 SEGMENTACIÓN DE LA IMÁGENES

En el diagrama de flujo representado en la Figura 2.5, se evidencia el procedimiento de segmentación de imágenes. Para llevar a cabo este proceso, se realizó una adaptación al código presentado en la sección anterior. En dicha adaptación, se emplearon bibliotecas específicas, la función show_anns, la carga del modelo y el generador de máscaras.

Las imágenes de la base de datos se almacenaron en una carpeta denominada "bandej". Posteriormente, se creó una nueva carpeta llamada “bandeja” para reubicar todas las imágenes con nombres modificados. Con el propósito de facilitar la manipulación, se les asignaron nombres numéricos del 1 al 1.027. Asimismo, se crearon dos carpetas adicionales: uno titulado “Segmentos” para almacenar las carpetas de los segmentos correspondiente a cada una de las imágenes y otro “Matrices” para almacenar las matrices correspondientes a los segmentos de cada una de las imágenes.

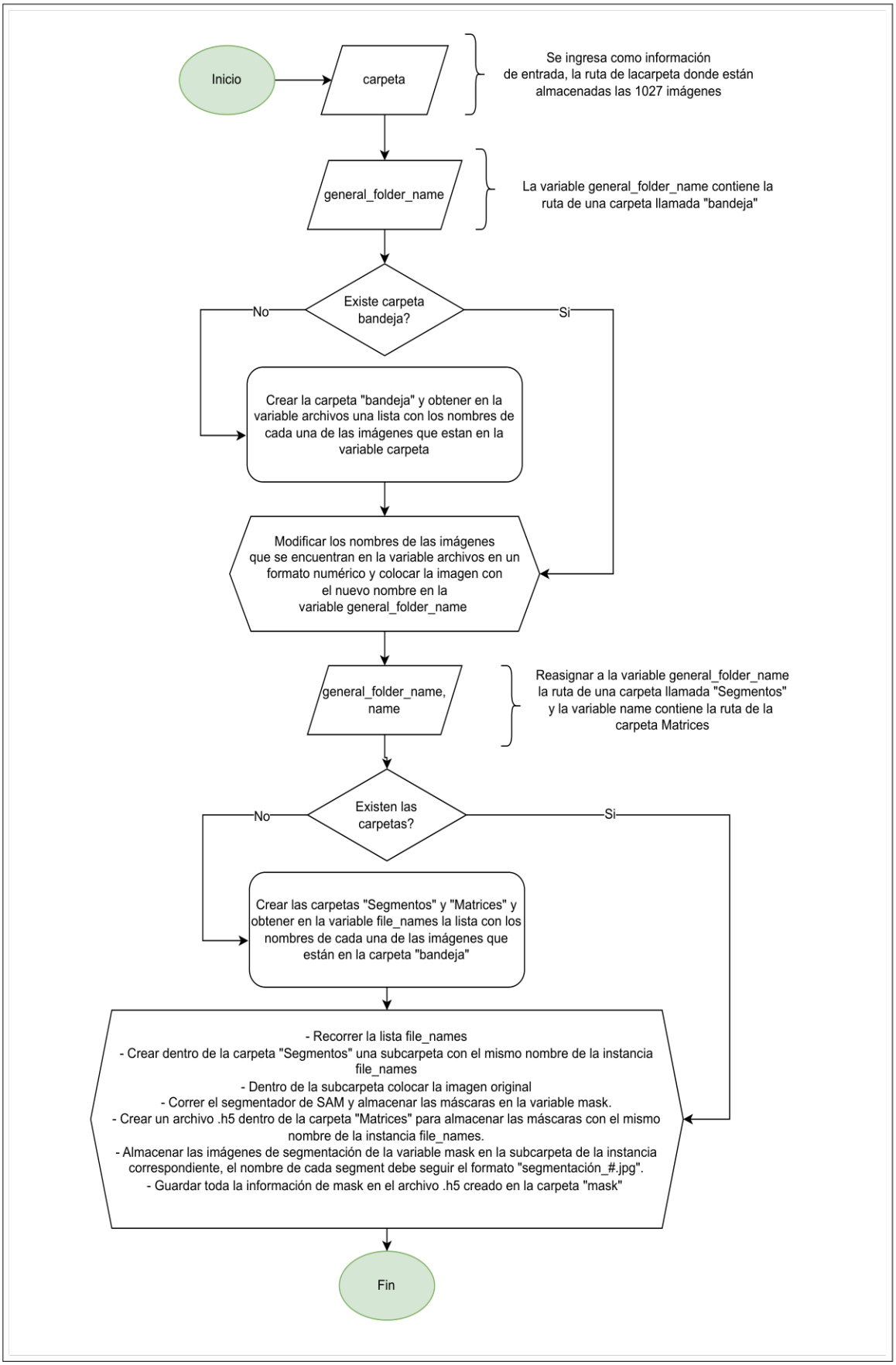

Figura 2.5: Diagrama de flujo del scrip para el proceso de las 1,027 imágenes.

El código desarrollado para la segmentación de las imágenes (que comienza a partir del Código 2.10) presentadas más adelante en esta misma sección posibilita recorrer la totalidad de la carpeta "bandeja", es decir, procesa cada imagen. Dentro de la carpeta “Segmentos”, se genera una subcarpeta con el mismo nombre de cada imagen original. En esta subcarpeta se ubica la imagen original y se van almacenando las imágenes resultantes de la segmentación. El formato de nombre para estas imágenes segmentadas es "segmentacion_#"donde # son números del 1 hasta el numero de segmentos que genera el segmentador para dicha imagen.

Por ejemplo, el código seleccionará la primera imagen de la carpeta “Segmentos”, que en este caso es la imagen denominada 1.jpg. Posteriormente, se creará una subcarpeta con el mismo nombre en la carpeta Segmentos correspondiente al archivo que está siendo procesado. Es decir, se generará una subcarpeta con el nombre "1". Dentro de esta subcarpeta, se colocará la imagen original (1.jpg) y se irán agregando las imágenes resultantes de la segmentación. Este mismo proceso se lo realiza con todas las imágenes.

Por otro lado como consecuencia de la segmentación de cada imagen, se genera un archivo .json que puede ser cargado en Python como un diccionario como lo muestra el Código 2.9.

```
annotation {
    "id"                : int,
    "segmentation"      : dict,
    "bbox"              : [x, y, w, h],
    "area"              : int,
    "predicted_iou"     : float,
    "stability_score"   : float,
    "crop_box"          : [x, y, w, h],
    "point_coords"      : [[x, y]],
}
```

Código 2.9: Diccionario de la segmentación de cada imagen.

Del Código 2.9 el “id” indica el id de segmentación, “segmentation” es la máscara guardada en formato COCO RLE[4] es aquí donde se almacena la matriz que se usa para recuperar

[4] **Coco RLE:** Es un formato que se refiere a la codificación de longitud de ejecución utilizada en el conjunto de datos COCO (Common Objects in Context), se utiliza para representar máscaras de segmentación de objetos. El formato RLE codifica la información de manera más eficiente [29].

cada segmento de la imagen, “bbox” es el cuadro alrededor de la máscara, en formato X, Y, W, H, “area” es el área en píxeles de la máscara, “predicted_iou” es a propia predicción del modelo sobre la calidad de la máscara, “stability_score” es una medida de la calidad de la máscara, “crop_box” es el recorte de la imagen utilizada para generar la máscara, en formato X, Y, W, H y “point_coords” es el punto coordina la entrada al modelo para generar la máscara.

Una vez que se completa la segmentación y se guardan todas las imágenes resultantes en la carpeta Segmentación para cada imagen, el código procede a almacenar todos los diccionarios de segmentación correspondientes a una imagen en un archivo con extensión .h5. Finalmente, estos archivos se colocan en la carpeta "Matrices". Es decir, para cada imagen procesada, se genera un archivo .h5 que contiene todos los datos de los segmentos creados. Cada archivo .h5 recibe el mismo nombre que la imagen en la que se está trabajando; por ejemplo, si se está procesando la imagen 100.jpg, el archivo resultante se denominará 100.h5.

A continuación, se presenta el código empleado para realizar el proceso de segmentación descrito anteriormente. Es importante señalar que se realizaron algunas modificaciones al código mencionado en la Sección 2.1, dado que el proceso de segmentación se llevó a cabo localmente y no en Google Colab, ya que debido a la cantidad de datos que hay que procesar el tiempo de ejecución de Google Colab es limitado.

El Código 2.10 instala las bibliotecas y dependencias necesarias para ejecutar el modelo de segmentación de imágenes proporcionado por Facebook Research [18], además de algunas bibliotecas adicionales comúnmente utilizadas en el campo de la visión por computadora y el aprendizaje profundo.

```
1 !pip install git+https://github.com/facebookresearch/segment-anything.git
2 pip install opencv-python pycocotools matplotlib onnxruntime onnx
3 !pip install torch
4 !pip install torchvision
5 !pip install h5py
```

Código 2.10: Proceso de segmentación 1.

El Código 2.11 configura el entorno para realizar: operaciones matemáticas y manipulación de datos mediante NumPy; permite la construcción y entrenamiento de modelos de aprendizaje profundo utilizando PyTorch; permite la generación de gráficos y visualizacio-

nes mediante Matplotlib; procesamiento de imágenes y visión por computadora a través de OpenCV; permite la manipulación del sistema de archivos mediante el módulo os y permite el trabajo con conjuntos de datos en formato HDF5 utilizando h5py.

Después se corre el Código 2.3 de la función *show_anns* que se la explicó en la sección 2.1 para mostrar las anotaciones (annotations) de objetos en una imagen.

```
1 import numpy as np
2 import torch
3 import matplotlib.pyplot as plt
4 import cv2
5 import torchvision
6 import os
7 import h5py
```

Código 2.11: Proceso de segmentación 2.

Como se observa en el Código 2.6, el Código 2.12 carga el modelo de segmentación SAM preentrenado, configura el entorno de ejecución (por ejemplo, el dispositivo), y crea un generador automático de máscaras que utiliza el modelo para segmentar imágenes automáticamente. El cambio que se hace en comparación con el Código 2.6 es en la línea 9 en la variable *device* colocándole cpu ya que se está usando una instancia (máquina virtual) de un servidor cuyas características se detallan en la Tabla 2.1.

```
1 #Modelo
2 import sys
3 sys.path.append("..")
4 from segment_anything import sam_model_registry, SamAutomaticMaskGenerator,
      SamPredictor
5
6 sam_checkpoint = "sam_vit_h_4b8939.pth"
7 model_type = "vit_h"
8
9 device = "cpu"
10
11 sam = sam_model_registry[model_type](checkpoint=sam_checkpoint)
12 sam.to(device=device)
13
14 mask_generator = SamAutomaticMaskGenerator(sam)
```

Código 2.12: Proceso de segmentación 3.

Tabla 2.1: Características de la instancia usada para el proceso de segmentación de las imágenes.

Configuración de hardaware	
CPU	6 vCPUs
Memoria	30.5 GB
Disco duro	450 GB
Tarjeta de video	16 MB

El Código 2.13 ofrece configuraciones para la generación automática de máscaras mediante un modelo denominado *"sam"* que corresponde al modelo SAM hablado anteriormente, con parámetros particulares para controlar la densidad de puntos, la calidad de la máscara, la estabilidad, el recorte de capas y puntos, y el filtrado de regiones pequeñas. La interpretación exacta está sujeta a la implementación concreta de SamAutomaticMaskGenerator y el modelo SAM.

```
1 mask_generator_2 = SamAutomaticMaskGenerator(
2     model=sam,
3     points_per_side=32,
4     pred_iou_thresh=0.86,
5     stability_score_thresh=0.92,
6     crop_n_layers=1,
7     crop_n_points_downscale_factor=2,
8     min_mask_region_area=100,  # Requires open-cv to run post-processing
9 )
```

Código 2.13: Proceso de segmentación 4.

En la línea 2 del Código 2.14 se define una variable carpeta que contiene la ruta al directorio que contiene las imágenes de comida de la base de datos original. En la línea 2 del Código 2.15 se crea una variable *general_folder_name* con el valor bandeja donde se almacenará las imágenes con los nuevos nombres, para esto primero comprueba si la carpeta existe, si no es así se crea dicha carpeta como lo indica la linea 3 y 4 del código. En la línea 7 en la variable archivos se almacena la lista de nombres de las imágenes en la variable carpeta.

```
1 # Ruta de la carpeta donde se encuentran las imánes de comida de la base original
2 carpeta = "bandej"
3
4 # Crear la carpeta "bandeja" si no existe, en esta carpeta se almacena las
    imagenes
```

Código 2.14: Proceso de segmentación 5.

```
# con los nuevos nombres
general_folder_name = "bandeja"
if not os.path.exists(general_folder_name):
    os.makedirs(general_folder_name)

# Obtener la lista de archivos en la carpeta
archivos = os.listdir(carpeta)
```

Código 2.15: Proceso de segmentación 6.

El Código 2.16 recorre la lista de la variable archivos dentro de la carpeta bandej, asegura de que cada elemento sea un archivo; en caso de que se cumpla esta condición, modifica el nombre del archivo conforme a una convención particular, que implica añadir un número y la extensión .jpg empezando por el número 1 y se va sumando 1. Como resultado, los archivos en la carpeta original (bandej) experimentan un cambio de nombre y se trasladan a una nueva carpeta denominada "bandeja".

```
# Cambio los nombres de los archivos
for i, nombre_archivo in enumerate(archivos):
    # Verifica si el elemento es un archivo (no un directorio)
    if os.path.isfile(os.path.join(carpeta, nombre_archivo)):
        # Cambiar el nombre del archivo empezando por 1 y se va sumando 1
        nuevo_nombre = f"{i + 1}.jpg"
        # Ruta completa del archivo original
        ruta_original = os.path.join(carpeta, nombre_archivo)
        # Ruta completa del archivo con el nuevo nombre
        ruta_nueva = os.path.join(general_folder_name, nuevo_nombre)
        # Renombrar el archivo
        os.rename(ruta_original, ruta_nueva)

print("Nombres de archivos cambiados correctamente.")
```

Código 2.16: Proceso de segmentación 7.

En el Código 2.17 y Código 2.18 se puede ver que:

- De la línea 1 hasta la línea 9 se verifica si las carpetas Segmentos y Matrices existen y si es que no se crean usando *os.makedis.*

- En la línea 12 y 13 se obtiene la lista de nombres de las imágenes en la carpeta *"bandeja".*

- De la línea 16 a la 19 se itera sobre cada imagen en la carpeta "bandeja", se lee la imagen utilizando OpenCV, y se convierte a formato RGB.

- De la línea 22 a la 24 se crea una carpeta en Segmentos para la imagen actual si aún no existe.

- En la línea 27 y 28 la imagen original se guarda en la carpeta correspondiente en Segmentos.

- En la línea 30 se genera una máscara para la imagen utilizando el generador automático de máscaras previamente definido *(mask_generator_2)* como se puede ver en el Código 2.13.

- En la línea 34 se genera el nombre del archivo HDF5 para cada imagen concatenando el nombre del archivo original *(file_name)* sin la extensión (los últimos 4 caracteres) con la extensión ".h5".

- En la línea 37 se crea la ruta completa del archivo HDF5 concatenando la carpeta de destino name (Matrices) con el nombre del archivo HDF5 generado anteriormente en la línea 34.

- En la línea 39 se abre el archivo HDF5 en modo de escritura utilizando el administrador de contexto with.

- En la línea 40 se itera sobre cada máscara generada y se accede a la información relacionada con la máscara a través de la variable *obj.*

- En las líneas 46, 49 y 52 se crea una imagen en color *(imagen_segmentada)* del mismo tamaño que la matriz de segmentación (segmentation). Se obtienen los colores correspondientes de la imagen original (image_rgb) utilizando la máscara de segmentación.

- En las líneas 55 y 56 se guarda la imagen segmentada en la carpeta correspondiente con un nombre específico.

- De la línea 60 hasta la línea 67 se crea un grupo en el archivo HDF5 con el nombre de la imagen segmentada. Dentro de este grupo, se crean conjuntos de datos (utilizando

create_dataset) para almacenar información específica relacionada con la máscara, tal como se detalla en el Código 2.9. Esto incluye la matriz de segmentación, el área, la caja delimitadora, el índice de superposición predicho, las coordenadas de los puntos, la puntuación de estabilidad y la caja de recorte.

```
# Crear la carpeta "Segmentos" si no existe
general_folder_name = "Segmentos"
if not os.path.exists(general_folder_name):
    os.makedirs(general_folder_name)

# Crear la carpeta "Matrices" si no existe
name = "Matrices"
if not os.path.exists(name):
    os.makedirs(name)

# Obtener la lista de nombres de archivos en la carpeta "bandeja"
folder_path = 'bandeja'
file_names = os.listdir(folder_path)

# Leer cada imagen, convertirla a formato RGB y agregarla a la lista
for file_name in file_names:
    file_path = os.path.join(folder_path, file_name)
    image = cv2.imread(file_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Crear la carpeta para la imagen actual si no existe
    nombre_carpeta_imagen = os.path.join(general_folder_name, file_name[:-4])
    if not os.path.exists(nombre_carpeta_imagen):
            os.makedirs(nombre_carpeta_imagen)

    # Guardar la imagen original en la carpeta
    ruta_absoluta = os.path.abspath(os.path.join(nombre_carpeta_imagen, file_name
        ))
    cv2.imwrite(ruta_absoluta, cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR))

    mask = mask_generator_2.generate(image_rgb)

    # Se tiene la lista 'masks' como se menciona en la pregunta
    # Nombre del archivo .h5
    nombre_archivo = file_name[:-4] +'.h5'
```

Código 2.17: Proceso de segmentación 8.

```
36      # Crear el archivo HDF5 en la carpeta de destino
37      ruta_archivo_h5 = os.path.join(name, nombre_archivo)
38
39      with h5py.File(ruta_archivo_h5, 'w') as hf:
40          for idx, obj in enumerate(mask):
41          # Obtener la matriz de segmentación de la máscara
42              segmentation = obj['segmentation']
43
44
45              # Crear una imagen en color del mismo tamaño que la matriz de
                    segmentación
46              imagen_segmentada = np.zeros(segmentation.shape + (3,), dtype=np.
                    uint8)
47
48              # Obtener los colores correspondientes de la imagen original
49              colores_originales = image_rgb[segmentation]
50
51              # Establecer los píxeles correspondientes a la segmentación con los
                    colores originales
52              imagen_segmentada[segmentation] = colores_originales
53
54              # Guardar la imagen segmentada en la carpeta correspondiente
55              nombre_imagen_segmentada = f"segmentacion_{idx + 1}.jpg"
56              cv2.imwrite(os.path.join(nombre_carpeta_imagen,
                    nombre_imagen_segmentada), cv2.cvtColor(imagen_segmentada, cv2.
                    COLOR_RGB2BGR))
57
58
59              #ARCHIVO
60              group = hf.create_group(nombre_imagen_segmentada[:-4])
61              group.create_dataset('segmentation', data=bool_array_to_uint8(obj['
                    segmentation']))
62              group.create_dataset('area', data=obj['area'])
63              group.create_dataset('bbox', data=obj['bbox'])
64              group.create_dataset('predicted_iou', data=obj['predicted_iou'])
65              group.create_dataset('point_coords', data=obj['point_coords'])
66              group.create_dataset('stability_score', data=obj['stability_score'])
67              group.create_dataset('crop_box', data=obj['crop_box'])
```

Código 2.18: Continuación de proceso de segmentación 8.

2.3 ELIMINACIÓN DE SEGMENTOS BASURA

En la Sección 2.1.1, se evidencia que, tras el proceso de segmentación, se genera una carpeta denominada Segmentos. En el interior de esta carpeta, se encuentran subcarpetas numeradas del 1 al 1027. Cada una de estas subcarpetas alberga la imagen original, cuyo nombre coincide con el número de la subcarpeta que la contiene. Además, contiene las imágenes correspondientes a la segmentación, cuyos nombres siguen el formato segmentación_#, donde # representa el número correspondiente al segmento.

En la Figura 2.6 se presenta un ejemplo de la organización de carpetas donde se observa que se inicia dentro de la carpeta Segmentos. Dentro de esta, se accede a la subcarpeta numerada como 4, la cual contiene 106 elementos. Entre estos elementos se encuentra la imagen original, cuyo nombre es 4, en correspondencia con la misma numeración de la subcarpeta que la contiene. Los demás archivos corresponden a las segmentaciones derivadas del archivo original.

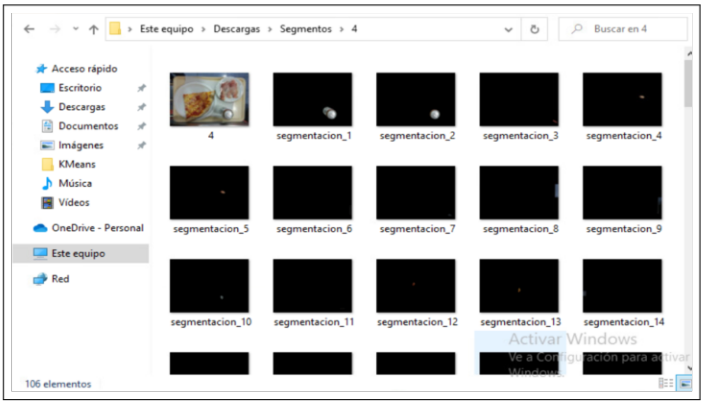

Figura 2.6: Archivos que contiene la subcarpeta 4.

El segmentador SAM produce imágenes correspondientes a cada objeto detectado por el algoritmo en la imagen. No obstante, no todas las imágenes de segmentos generadas son necesariamente válidas, ya que su utilidad está vinculada al objetivo específico de la segmentación. En el contexto de este trabajo, no se emplearán todas las imágenes de segmentos generadas, ya que la mayoría de ellas son consideradas imágenes no relevantes o basura. Por ende, se seleccionarán únicamente aquellas imágenes que exhiban los elementos presentes en la imagen original.

2.3.1 CRITERIOS DE ELIMINACIÓN DE IMÁGENES DE SEGMENTACIÓN

En este paso, es crucial señalar que la segmentación constituye un procedimiento destinado a separar los distintos elementos presentes en una imagen. Por ejemplo, consideremos una imagen de un plato de comida; mediante la segmentación, se generará una imagen para

cada componente presente en el plato. En otras palabras, si la imagen inicial muestra una bandeja con varios platos como se ve en la Figura 2.7, se crearán imágenes individuales que representan únicamente a los elementos que tiene cada plato como se lo puede ver en la Figura 2.8. No obstante, es importante destacar que en este proceso también se generarán imágenes de segmentación no deseadas, como aquellas que muestran únicamente el plato o la bandeja como se puede ver en la Figura 2.9. Estas imágenes carecen de utilidad para los objetivos específicos de este proyecto, razón por la cual deben ser eliminadas.

Figura 2.7: Imagen de bandeja original.

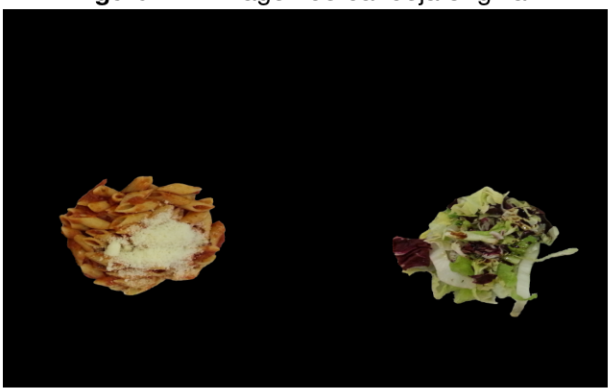

Figura 2.8: Segmentaciones válidas.

Figura 2.9: Segmentaciones no válidas.

Con lo dicho anteriormente, a continuación se detallan algunas consideraciones que se han tenido en cuenta para clasificar ciertas imágenes como basura de segmentación. Es im-

portante mencionar que este proceso se realizó manualmente con las 1027 imágenes, ya que no se encontró un algoritmo que pudiera optimizarlo. Una de las razones para optar por el enfoque manual es que no se ha hallado un algoritmo que incorpore todas las consideraciones necesarias para catalogar una imagen como basura. Además, incluso si existiera dicho algoritmo, siempre sería necesario revisar manualmente el proceso para asegurar su precisión.

- No se consideraron las imágenes en las cuales no se visualicen en su totalidad o parcialmente todos los elementos presentes en la imagen original. Un ejemplo de esto se observa en la Figura 2.8, donde se representan completamente dos elementos que forman parte de la imagen original.

- No se consideraron las imágenes de segmentación que no correspondan a alimentos o frutas, tales como botellas, vasos, endulzantes, caramelos, cubiertos, entre otros.

- No se consideraron las imágenes de segmentación en las cuales se observen platos o bandejas con huecos, como se muestra en la imagen de la Figura 2.9.

- Existen imágenes que contienen arroz, fideos, pasta, papas fritas, ensaladas, menestras, entre otros elementos. Al generar las imágenes de segmentación, se crea una imagen para cada fideo, cada papa frita, cada grano de arroz o cada grano de menestra, como se muestra en la Figura 2.10. No se consideraron todas estas imágenes individuales; únicamente se conserva la imagen de segmentación que represente la totalidad de la porción de arroz, la totalidad de la porción de fideos, etc.

Figura 2.10: Segmentación que se generan a partir de la Figura 2.7, esta imagen debería ser eliminada.

2.3.2 ELIMINACIÓN DE LOS GRUPOS EN LOS ARCHIVOS HDF5

Conforme se eliminan las imágenes de segmentación no deseadas, siguiendo los criterios establecidos en la Sección 2.2.1, se genera simultáneamente un archivo Excel (llamado Segmentos_paraEliminarMatrices) detallado en el Anexo 5.1.1. Este archivo registra las imágenes de segmentación que no son eliminadas, con el propósito de utilizarlo más adelante al integrar el código que automatiza la eliminación de los grupos en el archivo HDF5 correspondiente. Como resultado, en el archivo .h5 quedan únicamente los grupos correspondientes a los segmentos que no fueron eliminados.

Es importante tener presente que en el Código 2.17 se especificó la generación de un archivo .h5 por cada imagen (es decir habrán 1027 archivos .h5). En otras palabras, cada archivo .h5 contendrá la información de cada uno de los segmentos generados. Como se detalla en el Código 2.17 en las líneas 60 y 61, dentro de cada archivo .h5 se crean grupos para distinguir el contenido de cada segmento. Cada grupo recibe el mismo nombre que el segmento correspondiente; por ejemplo, si la imagen del segmento se denomina segmentación_6, el grupo que alberga la información del segmento llevará el mismo nombre: segmentación_6. Esta práctica se realiza con el objetivo de facilitar la eliminación de los grupos asociados a los segmentos que fueron eliminados en cada imagen.

Con el objetivo de eliminar los segmentos no deseados, se ha desarrollado un código que automatiza este proceso. Para llevar a cabo la ejecución de este código, se requiere tener los 1027 archivos .h5 almacenados en una carpeta específica.

En el Código 2.19 en la línea 2 se establece la ruta del archivo xlsx que contiene la información sobre los segmentos que no fueron eliminados. En este caso, el nombre del archivo es EliminarMatrices.xlsx. Las demás líneas del código muestra el archivo EliminarMatrices.xlsx cargado en un dataframe.

```
1 # Especifica la ruta del archivo xlsx en donde está la información de los
      segmentos que no fueron eliminados
2 ruta_archivo = 'EliminarMatrices.xlsx'
3
4 # Carga el archivo en un DataFrame de pandas
5 df = pd.read_excel(ruta_archivo)
6
7 # Muestra las primeras filas del DataFrame
8 df.head()
```

Código 2.19: Proceso de eliminación de los segmentos en los archivos .h5.

El Código 2.20 modifica ciertas columnas del DataFrame df al rellenar los valores nulos con cero y convertir los datos a tipo entero, y luego muestra las primeras 100 filas del DataFrame transformado.

```
1  # Transforma las columnas Unnamed 2 a Unnamed 9 en enteros y reemplaza NaN con
       cero
2  columnas_a_transformar = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8']
3
4  df[columnas_a_transformar] = df[columnas_a_transformar].fillna(0).astype(int)
5  df.head(100)
```

Código 2.20: Proceso de eliminación de los segmentos en los archivos .h5.

En el Código 2.21 y en el Código 2.22 en la línea 2 se define la ruta de la carpeta llamada Matrices que contiene los archivos .h5. En la línea 5 utiliza un bucle for para iterar a través de los archivos en la carpeta Matrices. En la línea 6 verifica que el archivo actual tenga la extensión .h5 antes de procesarlo. En las línea 7 y 10 concatena la ruta de la carpeta con el nombre del archivo y extrae el nombre del archivo sin la extensión. En la línea 13 filtra el DataFrame df para obtener las filas donde la columna id_subcarpeta coincide con el nombre del archivo. En la línea 15 comprueba si hay coincidencias para el archivo actual. En la linea 17 abre el archivo HDF5 en modo de lectura y escritura. En la línea 22 itera a través de los grupos (claves) dentro del archivo HDF5. En la línea 23 se verifica si la clave del grupo comienza con segmentacion_. En la línea 28 se elimina el grupo si cumple con las condiciones especificadas en el DataFrame. Por último en las líneas 30 y 33 se imprime un mensaje si se eliminó el grupo o no.

```
1  # Ruta a la carpeta Matrices, en esta carpeta se almacena las 1027 archivos .h5
2  carpeta_matrices = 'Matrices/'
3
4  # Iterar a través de los archivos .h5 en la carpeta Matrices
5  for archivo_h5 in os.listdir(carpeta_matrices):
6      if archivo_h5.endswith('.h5'):
7          ruta_completa = os.path.join(carpeta_matrices, archivo_h5)
8
9          # Obtener el nombre del archivo sin la extensión
10         nombre_archivo = os.path.splitext(archivo_h5)[0]
11
12         # Buscar coincidencias en el DataFrame usando id_subcarpeta
13         coincidencias = df[df['id_subcarpeta'] == int(nombre_archivo)]
```

Código 2.21: Proceso de eliminación de los segmentos en los archivos .h5.

```
        if not coincidencias.empty:
            # Abrir el archivo HDF5
            with h5py.File(ruta_completa, 'r+') as hf:
                # Obtener las claves (nombres de los grupos) dentro del archivo
                    HDF5
                group_keys = list(hf.keys())

                # Iterar a través de los grupos y eliminar los no deseados
                for key in group_keys:
                    if key.startswith('segmentacion_'):
                        segmentacion_id = int(key.split('_')[1])

                        # Verificar si el segmento debe ser eliminado
                        if segmentacion_id not in coincidencias['V1'].values and
                            segmentacion_id not in coincidencias['V2'].values and
                             segmentacion_id not in coincidencias['V3'].values
                            and segmentacion_id not in coincidencias['V4'].values
                             and segmentacion_id not in coincidencias['V5'].
                            values and segmentacion_id not in coincidencias['V6'
                            ].values and segmentacion_id not in coincidencias['V7
                            '].values and segmentacion_id not in coincidencias['
                            V8'].values:
                            del hf[key]

                print(f"Grupos no deseados eliminados para {archivo_h5}")

        else:
            print(f"No se encontraron coincidencias para {archivo_h5}")
```

Código 2.22: Continuación del proceso de eliminación de los segmentos en los archivos .h5.

2.4 ETIQUETAMIENTO DE LOS SEGMENTOS

Después de obtener las imágenes correspondientes a los segmentos de las 1027 imágenes, se procede a realizar el etiquetado de cada uno de estos segmentos. En este proceso, se utilizaron las etiquetas proporcionada por la base de datos original UNIMB 2016 [2]. No obstante, dado que esta base de datos se centra en la comida italiana y las etiquetas están redactadas en italiano, fue necesario realizar una adaptación al español para el presente trabajo. Además, se incorporaron algunas etiquetas adicionales que se consideraron ausentes en la base de datos original ya que cuando se realizó este proceso faltaban etiquetas a

ciertas imágenes, quedando finalmente 77 etiquetas como se puede ver en la Figura 3.1.

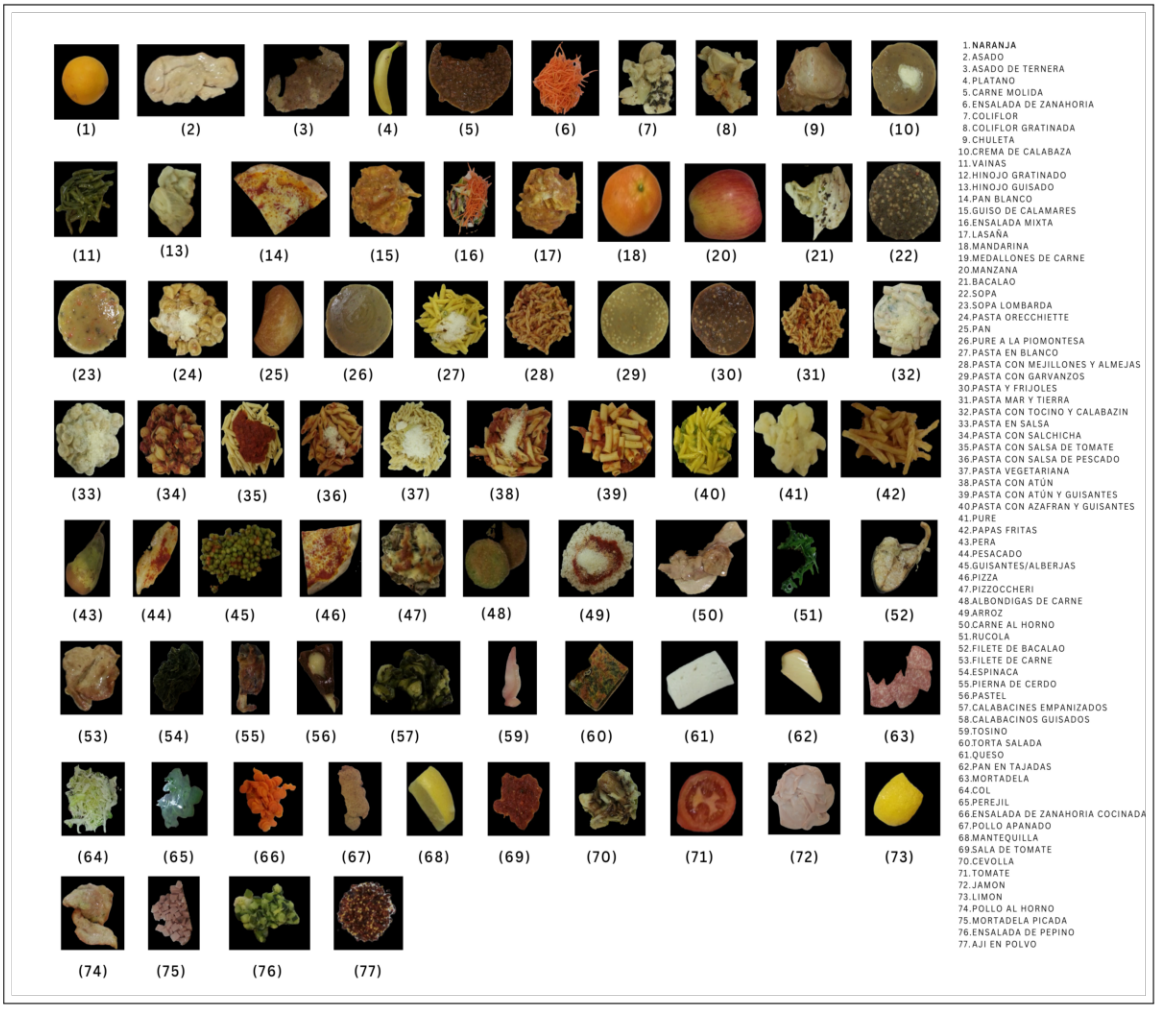

Figura 2.11: Imágenes segmentadas de las 77 categorías de alimentos en el conjunto de datos. A la derecha, los nombres de las clases. Hay que tener en cuenta que en algunos casos, los alimentos difieren ligeramente en los ingredientes.

Una vez que se obtienen las etiquetas con sus respectivos IDs, se procede a registrar el nombre de cada segmento en un archivo de Excel. En dicho archivo se detalla la información de la subcarpeta, el nombre de la imagen del segmento y el ID de la etiqueta correspondiente. Se busca asignar únicamente una etiqueta a cada segmento, aunque algunos segmentos presentan más de dos objetos, a los cuales se les asigna el número correspondiente de etiquetas. Es relevante señalar que ningún segmento cuenta con más de tres etiquetas. Este archivo Excel (llamado Etiquetamiento.xlsx) se encuentra disponible en el Anexo 5.2.1.

2.5 COMPRENSIÓN DE LOS ARCHIVOS .H5

Dada la extensa cantidad de información contenida en cada uno de los segmentos almacenados en los archivos .h5, estos presentan un considerable peso en la mayoría de los casos

pasan el 1GB. Con el objetivo de evitar el uso excesivo de recursos de hardware, se opta por almacenarlos en un formato comprimido mediante archivos .zip. Para llevar a cabo este proceso de compresión de manera automatizada, se implementa un código en Python.

El Código 2.23 importa el módulo os que proporciona una interfaz para manipulación de archivos y directorios; importa zipfile que permite trabajar con archivos zip incluyendo la creación, extracción y manipulación de archivos comprimidos; importa el módulo h5py que proporciona funcionalidades para trabajar con archivos en formato HDF5 e importa el módulo shutil, que ofrece operaciones de alto nivel para la manipulación de archivos y directorios, incluyendo operaciones de copia, movimiento y eliminación.

```
1 import os
2 import zipfile
3 import h5py
4 import os
5 import shutil
```

Código 2.23: Importa módulos para utilizar las funciones y clases que proporciona el código.

El Código 2.24 realiza lo siguiente:

- En las líneas 2 y 5 se definen dos rutas de carpetas: i) carpeta_original es la ruta de la carpeta que contiene los archivos con extensión .h5 y ii) carpeta_destino es la ruta de la carpeta donde se crearán los archivos comprimidos (.zip).
- En las líneas 8 y 9 se verifica si la carpeta de destino (carpeta_destino) existe. Si no existe, la crea utilizando os.makedirs .
- En la línea 12 se crea una lista llamada archivos_h5 que contiene los nombres de todos los archivos en la carpeta original (carpeta_original) que tienen la extensión .h5.
- La línea 15 itera sobre cada archivo en la lista archivos_h5.
- En las líneas 16 y 17 para cada archivo '.h5', se crea un nombre para el archivo comprimido (.zip) al cambiar la extensión del archivo .h5 a .zip.
- En las líneas 20 y 23 se construyen las rutas completas (ruta_h5 y ruta_zip) para el archivo .h5 original y el archivo comprimido, respectivamente.
- En las líneas 26 y 27 se utiliza la biblioteca zipfile para crear un archivo comprimido (.zip) en la carpeta de destino (carpeta_destino). Dentro del archivo comprimido,

agrega el archivo .h5 original. El archivo comprimido se crea con compresión usando el método ZIP_DEFLATED.

- La línea 29 imprime el proceso completado cuando todo el proceso de compresión ha terminado.

```
1  # Ruta de la carpeta original que contiene los archivos .h5
2  carpeta_original = 'Matrices'
3
4  # Ruta de la carpeta que se creará para almacenar las matrices comprimidas
5  carpeta_destino = 'Matrices_comprimidas'
6
7  # Crear la carpeta de destino si no existe
8  if not os.path.exists(carpeta_destino):
9      os.makedirs(carpeta_destino)
10
11 # Lista de archivos .h5 en la carpeta original
12 archivos_h5 = [archivo for archivo in os.listdir(carpeta_original) if archivo.
       endswith('.h5')]
13
14 # Iterar sobre cada archivo .h5
15 for archivo_h5 in archivos_h5:
16     # Crear el nombre de la carpeta .zip
17     nombre_zip = os.path.splitext(archivo_h5)[0] + '.zip'
18
19     # Ruta completa del archivo .h5
20     ruta_h5 = os.path.join(carpeta_original, archivo_h5)
21
22     # Ruta completa de la carpeta .zip
23     ruta_zip = os.path.join(carpeta_destino, nombre_zip)
24
25     # Crear la carpeta .zip y agregar el archivo .h5 comprimido
26     with zipfile.ZipFile(ruta_zip, 'w', compression=zipfile.ZIP_DEFLATED) as zipf
           :
27         zipf.write(ruta_h5, os.path.basename(ruta_h5))
28
29 print("Proceso completado.")
```

Código 2.24: Comprensión de los archivos .h5 en formato zip y los almacena en una carpeta designada para matrices comprimidas.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 RESULTADOS

En este capítulo se muestra la estructura final de la base de datos y a su vez se muestra los resultados de la prueba de predicción de la base en algoritmos de perdición.

3.1.1 ESTRUCTURA DE LA BASE DE DATOS

La base de datos final se encuentra en la carpeta Base_Imagenes_Comida en el One Drive cuyo enlace se encuentra en la parte del Anexo 5.2.1 y está estructurada de la siguiente manera:

- Existe un archivo Excel denominado Etiquetamiento, compuesto por dos libros. El primero, titulado Etiquetas, contiene las 77 etiquetas junto con sus respectivos IDs. El segundo libro, denominado Etiquetamiento, exhibe la información referente a la asignación de IDs a los segmentos de las 1027 imágenes.

- La carpeta Segmentos que contiene las 1027 subcarpetas con las respectivas imagenes y las correspondientes imágenes de segmento.

- La carpeta Matrices_comprimidas alberga los 1027 archivos .h5, los cuales se encuentran comprimidos debido a su peso, asociado a la cantidad de información que contienen.

- La carpeta Entrenamiento, que alberga todos los archivos necesarios para el entrenamiento del algoritmo, incluye varios elementos. En primer lugar, contiene una subcarpeta denominada Asado de carne, que contiene 106 imágenes correspondientes a esa categoría específica. Asimismo, cuenta con otra subcarpeta llamada Pan, que contiene 431 imágenes también relacionadas con dicha categoría. Además de las subcarpetas de imágenes, la carpeta de entrenamiento incluye tres archivos de texto. Etiquetas.txt contiene los nombres de las dos etiquetas utilizadas en el entrenamiento. Por otro lado, test.txt contiene las imágenes utilizadas para las pruebas del entrenamiento, mientras que train.txt enumera los nombres de las imágenes utilizadas para el entrenamiento, organizadas por categoría.

- Un documento en Word llamado GuíaUsuario en donde se podrá encontrar información de como usar la base de datos creada y algunos recomendaciones para su uso.

3.1.2 USO DE LA BASE DE DATOS CREADA PARA UN ENTRENAMIENTO

En el proceso de entrenamiento, se empleó un algoritmo de aprendizaje profundo. Es importante destacar que este algoritmo se encarga de entrenar las imágenes por categorías. Por ende, de las 77 categorías disponibles, se seleccionaron únicamente dos: i) Pan, que cuenta con 431 imágenes, y ii) Asado de carne, que tiene 106 imágenes como se puede ver en la Figura 3.1. Se eligieron solo dos categorías ya que eran las categorías con más datos, ideal para un correcto entrenamiento del algoritmo. En cuanto al entrenamiento, se seleccionó el 80 % de las imágenes para esta fase, reservando el 20 % restante para las pruebas.

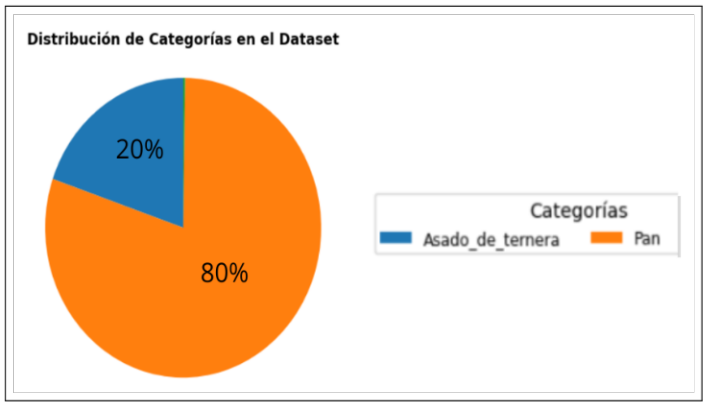

Figura 3.1: Diagrama de pastel de las categorías de imágenes de segmentos usadas en el proceso de entrenamiento.

A continuación, el algoritmo utilizado ingresa todas las imágenes destinadas al entrenamiento, y de entre estas selecciona algunas al azar, mostrándolas junto con sus respectivas etiquetas, como se ilustra en la Figura 3.2.

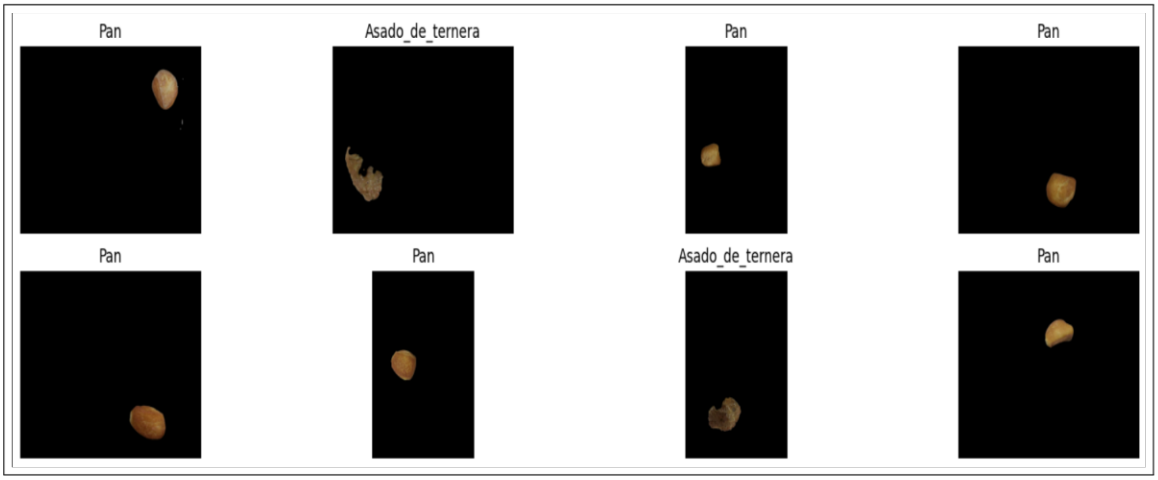

Figura 3.2: El subplot exhibe una selección aleatoria de imágenes tomadas de todas aquellas que se emplean en el proceso de entrenamiento, cada una acompañada de su etiqueta correspondiente.

En la Figura 3.3 se ilustra cómo el algoritmo aplica la estrategia de Data Augmentation[1] tanto en el conjunto de entrenamiento como en el de pruebas. Esto incluye rotación aleatoria, cambio de tamaño, recorte y volteo horizontal de las imágenes, todo enfocado específicamente en el conjunto de entrenamiento. Para el conjunto de pruebas, se utiliza el redimensionamiento de las imágenes para garantizar consistencia y calidad durante la evaluación del modelo.

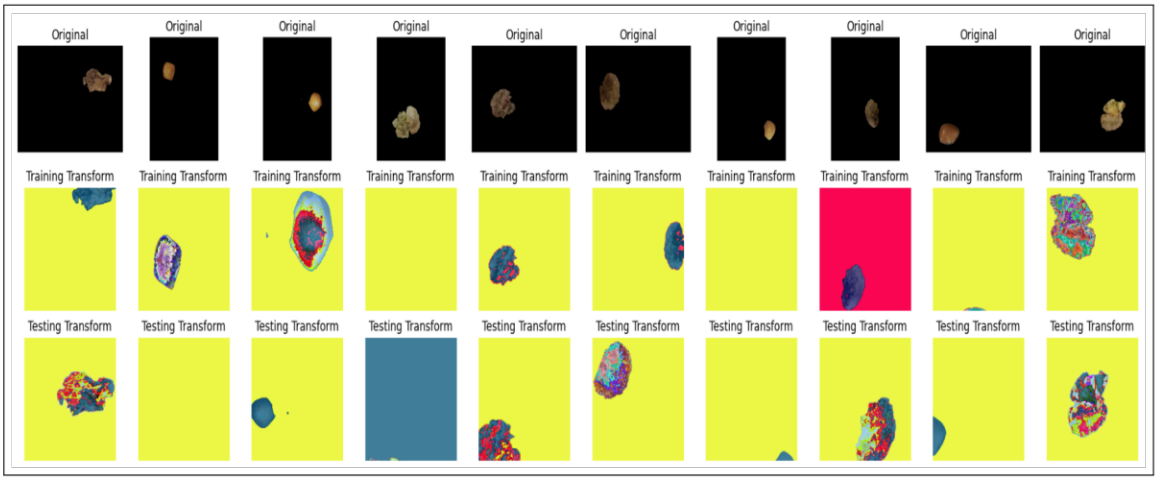

Figura 3.3: Preprocesamiento para el modelo de deep learning, que incluye la aplicación de la estrategia de Data Augmentation tanto en el conjunto de entrenamiento como en el de pruebas de las imágenes utilizadas para el entrenamiento del algoritmo.

Después de entrenar la imágenes en el algoritmo se tiene los resultados como se ve en la Figura 3.4. Este resultado muestra la primera época (Epoch) de un total de 10 épocas planificadas para el entrenamiento. La parte de Training Progress muestra el progreso del entrenamiento en términos de iteraciones. Se ejecutaron un total de 7 iteraciones. Esto indica que todas las iteraciones se completaron, mostrando el 100 %. El tiempo estimado para completar las iteraciones es de 26 segundos, con un promedio de 3.72 segundos por iteración. Para el Testing Progress muestra el progreso de la evaluación del modelo en términos de iteraciones. Se ejecutaron un total de 2 iteraciones para la evaluación del modelo. Esto indica que todas las iteraciones se completaron, mostrando el 100 %. El tiempo estimado para completar las iteraciones es de 4 segundos, con un promedio de 2.03 segundos por iteración. Además muestra un resumen de las métricas clave obtenidas durante la primera época:

[1] **Data Augmentation:** Es una estrategia para ampliar la variedad de información con la que se entrena, lo que favorece que los modelos adquieran conocimientos más diversos sobre distintos patrones y atributos, reduciendo así el riesgo de sobreajuste [30].

- ***train_loss y test_loss*** representan las pérdidas (errores) obtenidas durante el entrenamiento y la evaluación, respectivamente. Para este caso, el modelo tiene una pérdida de entrenamiento de aproximadamente 0.8824 y una pérdida de prueba de aproximadamente 0.2250.

- ***train_acc y test_acc*** representan las precisión (exactitud) obtenidas durante el entrenamiento y la evaluación, respectivamente. Para este caso, el modelo tiene una precisión de entrenamiento de aproximadamente 0.7342 y una precisión de prueba de aproximadamente 0.9689.

Basándose en los datos de la Figura 3.4, se puede concluir que el entrenamiento del modelo parece ser efectivo, dado que ha logrado alcanzar una alta precisión de casi el 97 % en el conjunto de prueba, pero presenta indicios significativos de sobreajuste en los datos de entrenamiento ya una diferencia significativa entre la precisión de entrenamiento y prueba puede indicar sobreajuste, especialmente si la precisión en el conjunto de prueba es considerablemente mayor que en el conjunto de entrenamiento. Por otro lado la diferencia entre las pérdidas de entrenamiento y prueba puede ser un indicio de sobreajuste si la pérdida en el conjunto de entrenamiento es considerablemente mayor que en el conjunto de prueba. Por esto se recomienda monitorear la tendencia de estas métricas a lo largo de múltiples epochs para obtener una evaluación más completa del rendimiento del modelo

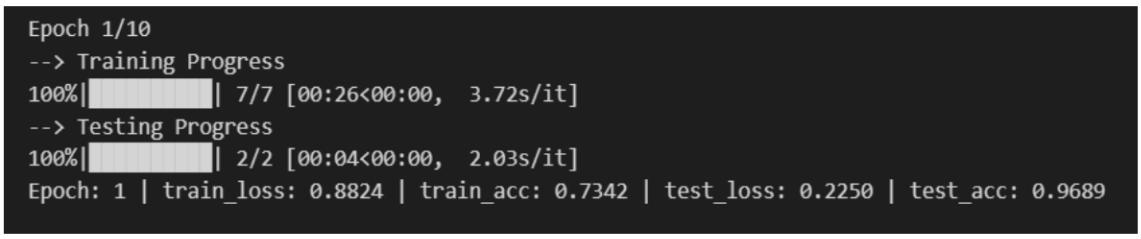

Figura 3.4: Progreso del entrenamiento durante la primera época (epoch) de entrenamiento del modelo.

En la Figura 3.5 muestra la precisión del modelo en el conjunto de datos de prueba, expresado como un porcentaje. En este caso, la precisión del modelo es del 97.20 %, lo que significa que el modelo clasificó correctamente el 97.20 % de las muestras en el conjunto de datos de prueba. Este resultado sugiere que el algoritmo ha aprendido eficazmente a generalizar patrones de los datos de entrenamiento a datos nuevos no vistos.

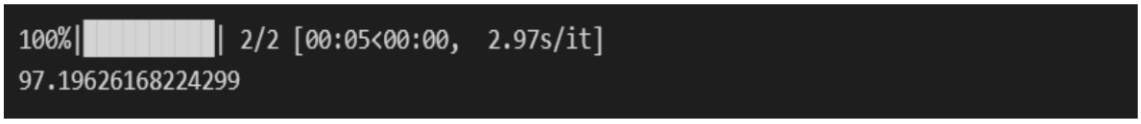

Figura 3.5: Resultado de la evaluación del modelo sobre el conjunto de datos de prueba.

Finalmente, en la Figura 3.6 se muestran los resultados de la predicción al ingresar dos imágenes descargadas de Internet que corresponden a las dos categorías utilizadas en el entrenamiento (Pan y Asado de carne). En el lado izquierdo de la Figura 3.6, se observa una imagen de Pan y el modelo la clasifica correctamente como Pan. En el lado derecho, se presenta otra imagen que el modelo identifica como Asado de carne. Estos resultados destacan el éxito del entrenamiento del modelo, evidenciando su capacidad para predecir con precisión las imágenes ingresadas. Es importante recordar que, dado que el modelo fue entrenado únicamente con dos categorías, solo puede predecir entre estas dos.

Figura 3.6: Resultado de la predicción de imágenes descargadas de la Internet.

3.2 CONCLUSIONES

- La creación de una base de datos de imágenes de comida puede abordarse de diversas maneras, ya sea utilizando imágenes propias, descargadas de Internet o incluso aprovechando bases de datos existentes, dependiendo de los objetivos específicos de la nueva base. Independientemente del método seleccionado para obtener las imágenes, es crucial seguir al menos los procesos de extracción, procesamiento y carga de la información. Sin embargo, al optar por utilizar un dataset existente, se evita el paso de procesamiento, como la limpieza de datos o la eliminación de imágenes duplicadas, ya que estas acciones suelen haber sido realizadas previamente en las imágenes de la base seleccionada. Por esta razón y dependiendo de los objetivos de la base, generalmente se prefiere utilizar una base existente y comenzar a trabajar en ella, porque generalmente la mayoría de imágenes en diferentes temáticas se las puede encontrar en Internet y así se evitaría gastar recursos como los que implicaría ir a tomar fotos propias.

- La segmentación en una base de datos de imágenes es de gran utilidad, ya que a través de ella es posible extraer características, realizar reconocimiento, clasificación, entre otras tareas, de los diversos objetos presentes en una imagen. Este proceso permite implementar aplicaciones que pueden aceptar una fotografía de un plato de comida, identificar los diferentes alimentos presentes en él e incluso proporcionar información sobre las categorías de cada componente del plato y las calorías totales del mismo.

- SAM es una de las inteligencias artificiales más potentes en segmentación, capaz de segmentar con una precisión notable cualquier imagen proporcionada. Lo notable radica en que esta IA ofrece la información de la segmentación tanto en forma de matrices o datos numéricos. Esta información, junto con un poco de programación, permite recuperar las imágenes de cada segmento, los cuales resultan sumamente útiles para el entrenamiento en diversos algoritmos. Tener disponible en la base de datos la información de las matrices de cada segmento es de gran utilidad para los algoritmos, ya que estos no necesitarían entrenarse con las imágenes de los segmentos, sino que basta con entrenarlos con cada una de las matrices. Aunque el algoritmo debe encargarse del proceso completo de transformación de imágenes, solo es necesario construir el algoritmo adecuado para que se adapte de manera óptima a una base de datos que proporciona matrices de segmentación.

- Aunque no existe una forma estandarizada de estructurar una base de datos de imágenes, es fundamental incluir toda la información relevante para su uso efectivo. En este trabajo, se optó por organizar la mayor parte de los datos en carpetas, que contienen tanto las imágenes originales como los segmentos, complementado con un archivo Excel que alberga las etiquetas de los segmentos. La elección de utilizar un archivo .xlsx se basó en sus funcionalidades, capacidad de interactividad y facilidad de uso, entre otras ventajas destacadas.

3.3 RECOMENDACIONES

- Para el proceso de extracción de imágenes, ya sea descargándolas de Internet o seleccionándolas de una base de datos existente, es recomendable establecer previamente las etiquetas o categorías a utilizar dependiendo del objetivo que vaya a tener la base de datos, pero comúnmente se selecciona categorías generales que puedan contener a otras categorías, se puede armar carpetas para que cuando se este des-

cargando las imágenes simplemente colocarlas dentro de su categoría correcta.Esto facilitará la selección de las imágenes pertinentes. En el caso de la descarga desde Internet, la definición de etiquetas guiará la búsqueda y selección de imágenes. Si se opta por una base de datos preexistente, se sugiere elegir una con un número adecuado de datos y con etiquetas ya definidas. Esto permitirá aprovechar las etiquetas existentes o ajustarlas según las necesidades específicas de la nueva base de datos a crear.

- Se sugiere contar con un equipo de personas para agilizar el proceso de etiquetado, dado que en general, la cantidad de datos suele ser demasiado grande para que una sola persona o un par de personas puedan manejarlo eficientemente. Alternativamente, se recomienda utilizar la plataforma de Amazon Mechanical Turk (MTurk) para obtener el etiquetado manual a través de Tareas de Inteligencia Humana (HITs).

- Antes de crear una base de datos, es fundamental definir claramente su propósito y los objetivos que debe cumplir, así como comprender cómo se utilizará. Esto ayuda a determinar qué datos deben incluirse en la base y cómo debe estructurarse. Además, se recomienda mantener un orden adecuado y una estructuración coherente durante todo el proceso de creación y almacenamiento de datos. La falta de claridad en este aspecto puede resultar en procesos prolongados y, en última instancia, en la no utilización de los resultados debido a una mala estructuración de la base de datos.

- Para optimizar el proceso de segmentación de imágenes, es crucial considerar el alto consumo de recursos computacionales que este conlleva. Para evitar que el proceso sea excesivamente prolongado, con el riesgo de llevar meses completar una sola tarea, se recomienda asegurar el acceso a recursos computacionales adecuados teniendo en cuenta el número de datos a procesar, ver la arquitectura de hadware (CPU o GPU), entre otras. Esto garantizará que la segmentación se realice de manera eficiente y sin demoras significativas.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] M. Mohri, A. Rostamizadeh y A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[2] G. Ciocca, P. Napoletano y R. Schettini, "Food recognition: a new dataset, experiments, and results," *IEEE journal of biomedical and health informatics*, vol. 21, n.º 3, págs. 588-598, 2016.

[3] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. Paritosh y L. M. Aroyo, ""Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI," en *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, págs. 1-15.

[4] J. P. Mueller y L. Massaron, *Machine learning for dummies*. John Wiley & Sons, 2021.

[5] T. Erl, W. Khattak y P. Buhler, *Big data fundamentals: concepts, drivers & techniques*. Prentice Hall Press, 2016.

[6] E. Bello, *Guía de Procesos ETL: Qué son, cómo usarlos y herramientas clave. IEBSchool*.

[7] Y. Zhang, J. Xin, X. Li y S. Huang, "Overview on routing and resource allocation based machine learning in optical networks," *Optical Fiber Technology*, vol. 60, dic. de 2020, ISSN: 10685200. DOI: 10.1016/j.yofte.2020.102355.

[8] S. Hou, Y. Feng y Z. Wang, "VegFru: A Domain-Specific Dataset for Fine-Grained Visual Categorization," en *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, págs. 541-549. DOI: 10.1109/ICCV.2017.66.

[9] P. Zhou, C. Bai, K. Ying, J. Xia y L. Huang, "RWMF: A Real-World Multimodal Foodlog Database," en *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, págs. 962-968. DOI: 10.1109/ICPR48806.2021.9412433.

[10] P. Kaur, K. Sikka, W. Wang, S. Belongie y A. Divakaran, "Foodx-251: a dataset for fine-grained food classification," *arXiv preprint arXiv:1907.06167*, 2019.

[11] C. C. Rivero Hechavarría, I. Pérez Pupo, P. Y. Piñero Pérez et al., "Proceso de limpieza de datos y extensión del Repositorio para Investigaciones en Gestión de Proyectos," 2018.

[12] M. F. Gallardo Albarracín, "Construcción de una base de datos para análisis del diseño de página de documentos digitales.," B.S. thesis, Quito, 2021, 2021.

[13] K. He, X. Zhang, S. Ren y J. Sun, “Deep residual learning for image recognition,” en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, págs. 770-778.

[14] S. V. Orea, A. S. Vargas y M. G. Alonso, “Minería de datos: predicción de la deserción escolar mediante el algoritmo de árboles de decisión y el algoritmo de los k vecinos más cercanos,” *Ene*, vol. 779, n.º 73, pág. 33, 2005.

[15] T. Gebru, J. Morgenstern, B. Vecchione et al., “Datasheets for datasets,” *Communications of the ACM*, vol. 64, n.º 12, págs. 86-92, 2021.

[16] B. Jähne, *Digital image processing*. Pearson, 2018.

[17] N. L. S. Palomino y U. N. R. Concha, “Técnicas de segmentación en procesamiento digital de imágenes,” *Revista de investigación de Sistemas e Informática*, vol. 6, n.º 2, págs. 9-16, 2009.

[18] A. Kirillov, E. Mintun, N. Ravi et al., “Segment Anything,” *arXiv:2304.02643*, 2023.

[19] A. Fernandez, *Python 3 al descubierto*. Alfaomega Grupo Editor, 2013.

[20] E. M. Rojas, “Machine Learning: análisis de lenguajes de programación y herramientas para desarrollo,” *Revista Ibérica de Sistemas e Tecnologias de Informação*, n.º E28, págs. 586-599, 2020.

[21] W. McKinney, *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. .º’Reilly Media, Inc.", 2012.

[22] E. Stevens, L. Antiga y T. Viehmann, *Deep learning with PyTorch*. Manning Publications, 2020.

[23] S. Brahmbhatt, *Practical OpenCV*. Apress, 2013.

[24] A. Dosovitskiy, L. Beyer, A. Kolesnikov et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.

[25] K. He, X. Chen, S. Xie, Y. Li, P. Dollár y R. Girshick, “Masked autoencoders are scalable vision learners,” en *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, págs. 16 000-16 009.

[26] A. Albrecht y F. Naumann, “Managing ETL Processes.,” *NTII*, vol. 8, págs. 12-15, 2008.

[27] M. von Rosing, S. White, F. Cummins y H. de Man, *Business Process Model and Notation-BPMN.* 2015.

[28] J. L. Lovaco Hernández, “Aceleración de algoritmos de localización para robots móviles en 3D mediante CUDA®,” Tesis de mtría., 2014.

[29] R. Khandelwal. “COCO and Pascal VOC data format for Object detection.” Consultado el 18/02/2024. (2019), dirección: https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5.

[30] V. Veríssimo y R. Costa, “Using Data Augmentation and Neural Networks to Improve the Emotion Analysis of Brazilian Portuguese Texts,” en *Proceedings of the Brazilian Symposium on Multimedia and the Web*, 2020, págs. 13-20.

5 ANEXOS

5.1 ANEXO I

5.1.1 REPOSITORIO DE GITHUB

En el siguiente enlace se encuentra los archivos de código usado en todo el proceso de la creación de la base de datos.

https://github.com/David3199san/Base_ImagenesComida.git

5.2 ANEXO II

5.2.1 BASE DE DATOS DE IMÁGENES

En el siguiente enlace se encuentra la base de datos final.

https://epnecuador-my.sharepoint.com/:f:/g/personal/david_santos_epn_edu_ec/EhGMjoZJdZZDoDUxQqU
oNzUhCThQ