ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

APLICACIONES DE MACHINE LEARNING PARA EL ANÁLISIS DE COMIDA

MODELOS DE APRENDIZAJE AUTOMÁTICO PARA IDENTIFICACIÓN DE COMIDA

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

FREDERIK LEONARDO PEREZ USHIÑA
frederik.perez@epn.edu.ec

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

DMQ, febrero 2024

CERTIFICACIONES

Yo, FREDERIK LEONARDO PEREZ USHIÑA 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.

FREDERIK LEONARDO PEREZ USHIÑA

Certifico que el presente trabajo de integración curricular fue desarrollado por FREDERIK LEONARDO PEREZ USHIÑA, 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.

FREDERIK LEONARDO PEREZ USHIÑA
MSC. FRANKLIN LEONEL SANCHEZ CATOTA
PhD. CHRISTIAN JOSÉ TIPANTUÑA TENELEMA

ÍNDICE DE CONTENIDO

RESUMEN

Esta tesis presenta un estudio integral sobre la aplicación de técnicas de aprendizaje automático (ML), con un enfoque en Redes Neuronales Convolucionales (CNN), para el análisis de datos alimenticios. En una era donde la salud dietética y la calidad de los alimentos son de suma importancia, la necesidad de métodos avanzados, precisos y eficientes para analizar y clasificar alimentos nunca ha sido más crítica. Este trabajo aprovecha las capacidades del ML para cerrar esta brecha, ofreciendo soluciones innovadoras para la clasificación de alimentos, el análisis nutricional y la evaluación de calidad.

Utilizando un rico conjunto de datos de Food 101 y empleando algoritmos de ML de última generación, esta investigación navega a través de los desafíos del reconocimiento de imágenes de alimentos, proporcionando perspectivas sobre los aspectos texturales, nutricionales y cualitativos de varios alimentos. La tesis no solo profundiza en las tecnicidades del desarrollo de algoritmos y la optimización de modelos, sino que también contextualiza los hallazgos dentro de las implicaciones más amplias para la industria alimentaria, el monitoreo de la salud y la conciencia del consumidor.

PALABRAS CLAVE: Aprendizaje Automático, Redes Neuronales Convolucionales, Análisis de Alimentos, Reconocimiento de Imágenes, Análisis Nutricional, Enfoques Basados en Datos

ABSTRACT

This thesis presents a comprehensive study on the application of machine learning (ML) techniques, with a focus on Convolutional Neural Networks (CNNs), for the analysis of food data. In an era where dietary health and food quality are of paramount concern, the need for advanced, accurate, and efficient methods to analyze and classify food items has never been more critical. This work leverages the capabilities of ML to bridge this gap, offering innovative solutions for food classification, nutritional analysis, and quality assessment. Utilizing a rich dataset from Food 101 and employing state-of-the-art ML algorithms, this research navigates through the challenges of food image recognition, providing insights into the textural, nutritional, and qualitative aspects of various food items. The thesis not only delves into the technicalities of algorithm development and model optimization but also contextualizes the findings within the broader implications for the food industry, health monitoring, and consumer awareness.

KEYWORDS: Machine Learning, Convolutional Neural Networks, Food Analysis, Image Recognition, Nutritional Analysis, Data-Driven Approaches

1 INTRODUCCIÓN

El reconocimiento automático de imágenes de comida es un campo de investigación que ha tomado relevancia en los últimos años dentro del área de procesamiento de imágenes y el aprendizaje automático (ML: Machine Learning) . El reconocimiento preciso de alimentos en imágenes tiene el potencial de mejorar aplicaciones relacionadas con la nutrición y salud, como el monitoreo de dietas, detección de alérgenos o sustitutos de ingredientes [1].

El ML, y en particular el aprendizaje profundo (DL: Deep Learning), han demostrado un gran potencial para extraer características visuales que permiten clasificar y reconocer imágenes de forma automatizada. Las técnicas de DL como las redes neuronales convolucionales (CNN: Convolutional Neural Network) han superado ampliamente el rendimiento de enfoques tradicionales basados en extracción manual de características [1] para la identificación de alimentos [3]. Sin embargo, a pesar de estos avances, el reconocimiento preciso de imágenes de comida sigue presentando desafíos únicos, como la alta variabilidad en apariencia, cocción e iluminación. Además, la disponibilidad de datasets etiquetados de alimentos es limitada en comparación con otros dominios de imágenes [4]. Por estas razones, se requiere investigación focalizada para desarrollar modelos de ML que aborden de forma efectiva estos retos específicos.

En el presente trabajo de integración curricular, se presenta una revisión de los fundamentos teóricos de ML para la tarea de reconocimiento automático de imágenes de comida. Se exploran técnicas de aprendizaje supervisado y no supervisado para implementar al menos dos algoritmos de ML.Para mejorar el desempeño de los modelos, se aplicarán técnicas de preprocesamiento y aumento de datos. Además, se seleccionarán métricas adecuadas para evaluar estos modelos de manera efectiva.

1.1 OBJETIVO GENERAL

Analizar modelos de ML para el reconocimiento de comida.

1.2 OBJETIVOS ESPECÍFICOS

1. Describir el fundamento teórico del uso de ML para el reconocimiento de imágenes de comida.

[1] **Extracción manual de características:** Implica el uso de algoritmos y técnicas específicas para identificar y cuantificar elementos visuales clave en las imágenes, como bordes, esquinas, texturas y colores, que son relevantes para la tarea de clasificación o reconocimiento de imágenes [2].

2. Revisar datasets de imágenes para el entrenamiento de modelos de ML en el reconocimiento de comida.
3. Revisar diferentes modelos de ML para el reconocimiento de imágenes de comida.
4. Aplicar los modelos de ML seleccionados para el reconocimiento de imágenes de comida considerando los datasets previamente seleccionados.
5. Analizar el comportamiento y los resultados de los modelos de ML utilizando métricas estándar u otras métricas que puedan ser definidas para evaluar su desempeño.

1.3 ALCANCE

El presente trabajo tiene como objetivo principal analizar al menos dos modelos de ML aplicados a la tarea de reconocimiento de imágenes de alimentos. Para lograr esto, se seguirá una metodología que consta de las siguientes etapas:

En primer lugar, se realizará una investigación bibliográfica sobre las técnicas y algoritmos de ML que se han utilizado para el reconocimiento de comida. Se analizarán los diferentes enfoques de ML, las técnicas y algoritmos empleados, así como los parámetros y variables de entrada y salida de cada modelo. Además, se revisarán las métricas más comunes utilizadas en la evaluación de los resultados de los modelos de ML. Todo ello permitirá establecer las bases teóricas necesarias para llevar a cabo el análisis de los modelos en el reconocimiento de comida.

Posteriormente, se recolectarán y prepararán conjuntos de datos (datasets) de imágenes de comida, provenientes de repositorios y bases de datos públicas. Estos datasets serán empleados para entrenar y evaluar los modelos seleccionados.

Una vez seleccionados los conjuntos de datos apropiados, se procederá con el preprocesamiento de las imágenes de comida. Esto abarcará desde la depuración de datos, eliminación de imágenes que no cumplan con los criterios de calidad, hasta el etiquetado adecuado de las imágenes y la adición de descripciones necesarias para su uso en algoritmos de aprendizaje supervisado y no supervisado.

Luego, se implementarán al menos dos modelos de ML diferentes, elegidos en base a las técnicas identificadas en la revisión de literatura. Se ajustarán los parámetros y se entrenarán los modelos utilizando los datasets de imágenes de comida previamente obtenidos. Finalmente, se presentará un cuadro comparativo con las características de cada modelo y su desempeño, a través de casos de uso o ejemplos.

1.4 MARCO TEÓRICO

1.4.1 INTRODUCCIÓN AL RECONOCIMIENTO DE IMÁGENES

1.4.1.1 Evolución de la visión por computadora y ML

La visión por computadora[2] y el ML[3] han experimentado una evolución significativa desde sus inicios, marcando importantes hitos en el desarrollo de la tecnología y la inteligencia artifical (AI: Artificial Intelligence). La visión por computadora emergió en la década de 1960, cuando los investigadores comenzaron a explorar cómo emular la visión humana a través de la computación. Durante este periodo, el enfoque se centraba en la percepción básica de objetos comunes, aunque con limitaciones en el reconocimiento de múltiples objetos y sus variaciones de forma [5]. En los años 30, elementos como la introducción del sistema de televisión y, en los años 50, el desarrollo del concepto de flujo óptico por James J. Gibson y la creación de la primera imagen digital por Russell Kirsch, sentaron bases importantes para el futuro de la visión artificial [6].

En los años 60, el desarrollo del algoritmo “nearest neighbor” permitió el reconocimiento básico de patrones en computadoras [7]. En cambio, en 1963, Lawrence Roberts, conocido como el padre de la visión artificial, publicó un trabajo significativo sobre la extracción de información tridimensional de objetos a partir de imágenes bidimensionales, marcando un hito en el desarrollo de esta tecnología. Durante los años 70, se introdujeron innovaciones como el primer robot inteligente con un sistema de visión y la invención de la matriz de filtros Bayer, que llevó las imágenes en color a la fotografía digital [6].

Un gran avance ocurrió en 2014 con la llegada de la era del DL. Los investigadores entrenaron ordenadores con millones de imágenes, utilizando la tecnología de DL, lo que demostró una superioridad sobre los algoritmos tradicionales. En 2016, el DL avanzó hasta lograr un procesamiento casi en tiempo real gracias a las CNN multicapa y el uso intensivo de datos y hardware moderno [5]. Para 2020, las CNN se habían convertido en el estándar de cálculo en la visión por computadora, con modelos de AI optimizados capaces de realizar tareas de visión por computadora en dispositivos de bajo coste. Estos desarrollos han permitido una precisión casi humana en muchas aplicaciones de visión por computadora [5].

---

[2] La **visión por computadora** se refiere al campo de la informática que trabaja en hacer que las computadoras “vean” y procesen imágenes y videos de manera similar a como lo hacen los humanos.

[3] **ML:** Es un subcampo de la inteligencia artificial que se centra en la creación de sistemas que pueden aprender de los datos.

La evolución de la visión por computadora y el ML desde sus inicios hasta la actualidad refleja un progreso extraordinario, pasando de la percepción básica de objetos a sistemas capaces de realizar tareas complejas con una eficiencia y precisión casi humana. Estos avances han allanado el camino para numerosas aplicaciones innovadoras en campos tan diversos como la medicina, la robótica, y el reconocimiento de imágenes de comida.

La detección y clasificación de objetos encuentra su aplicación en múltiples áreas [8], incluyendo:

- ❒ **Industria automotriz**: Para la detección de obstáculos y asistencia al conductor.
- ❒ **Medicina**: En el análisis de imágenes médicas y cirugías asistidas por robots.
- ❒ **Industria Alimentaria**: Para el control de calidad y clasificación de productos.
- ❒ **Logística**: En la identificación y seguimiento de paquetes.
- ❒ **Seguridad**: Para reconocimiento facial y vigilancia.

1.4.2 RELEVANCIA DEL RECONOCIMIENTO DE ALIMENTOS MEDIANTE ML

El reconocimiento de alimentos mediante ML tiene aplicaciones significativas en diversos sectores, incluyendo la industria alimentaria, agricultura, y salud pública:

- ❒ **Industria Alimentaria**: La AI junto con la visión por computadora, se utiliza para identificar defectos en los productos alimenticios, predecir la fragilidad de alimentos como la pasta seca, y controlar la calidad de los envases termosellados. Estas tecnologías también permiten la clasificación estética de productos y la determinación del punto de maduración de frutas y vegetales [9].
- ❒ **Agricultura**: El ML ayuda a determinar el momento óptimo para la recolección de productos agrícolas, considerando factores como la madurez, el precio de mercado y las condiciones climáticas [9].
- ❒ **Salud Pública**: Investigadores han utilizado ML para estudiar la relación entre la alimentación y la salud, como en el caso de las nueces y su relación con la reducción del riesgo de diabetes tipo 2 y enfermedades cardiovasculares [10].

La investigación en este campo sigue evolucionando, buscando no solo predecir problemas sino también generar nuevo conocimiento y recomendaciones para maximizar la vida útil

de los productos y desarrollar nuevos alimentos adaptados a las preferencias de los consumidores [9]. El reconocimiento de alimentos mediante ML es un área de gran impacto y potencial, que está transformando la manera en que producimos, inspeccionamos y entendemos los alimentos, con beneficios tangibles para la sociedad y la economía global.

1.4.3 ESTADO ACTUAL DEL RECONOCIMIENTO DE IMÁGENES DE COMIDA

Los avances recientes en el reconocimiento de imágenes de comida se centran principalmente en el uso de la AI y el DL. Por ejemplo, Facebook ha desarrollado un sistema que, basándose en fotos de comidas, es capaz de asociar estas imágenes con recetas preexistentes o una combinación de ellas. Este sistema se ha entrenado con una gran cantidad de casos resueltos y ahora puede identificar incluso ingredientes no visibles en la foto, como la sal y el azúcar [11].

La integración del reconocimiento de imágenes de comida con otras tecnologías está en constante evolución. Por ejemplo, Domino's Pizza colaboró con Dragontail Systems para desarrollar una cámara que verifica que cada pizza cumpla con los estándares de la compañía. Este tipo de tecnología, como la ofrecida por Agot.AI, se está implementando en varios nichos de la industria alimentaria, no solo en pizzerías, sino también en hamburgueserías, taquerías y otros tipos de restaurantes [12].

El reconocimiento de imágenes de comida ha avanzado significativamente gracias a la AI y el DL, con aplicaciones que van desde la mejora del control de calidad en restaurantes hasta la asistencia en la selección de frutas. Sin embargo, aún hay desafíos a superar, especialmente en términos de precisión y adaptabilidad a recetas y platos únicos.

1.4.4 FUNDAMENTOS TEÓRICOS DEL ML PARA EL RECONOCIMIENTO DE IMÁGENES DE COMIDA

1.4.4.1 Aprendizaje supervisado aplicado al reconocimiento de imágenes de comida

El aprendizaje supervisado es una técnica esencial en AI y ML, utilizada en áreas como el reconocimiento de imágenes de comida. Esta técnica implica el uso de datos etiquetados para entrenar modelos para realizar predicciones o clasificaciones [13]. El aprendizaje

supervisado utiliza datos etiquetados, mientras que el no supervisado busca patrones en datos sin etiquetar. Esta diferencia es crucial en términos de los tipos de problemas que cada uno puede resolver [14].

En el aprendizaje supervisado, los modelos se entrenan con ejemplos etiquetados. Cada ejemplo incluye una entrada (imagen de comida) y una etiqueta de salida (tipo de comida). El modelo aprende a asociar entradas con las salidas correctas para realizar predicciones o clasificaciones en nuevos datos [13].

Los algoritmos comunes en aprendizaje supervisado incluyen árboles de decisión, máquinas de vectores de soporte y redes neuronales. Las redes neuronales son particularmente efectivas en el reconocimiento de imágenes debido a su capacidad para identificar patrones en datos visuales [15]. Para lograr buenos resultados, es vital tener un conjunto de datos de entrenamiento de calidad y representativo, cubriendo adecuadamente las categorías de alimentos que se quieren reconocer [13], [14].

Una ventaja principal del aprendizaje supervisado es su precisión en tareas de clasificación y predicción. Sin embargo, enfrenta desafíos como la necesidad de un conjunto de datos etiquetados grande y el manejo de clases desequilibradas [14]. El rendimiento de un modelo de aprendizaje supervisado se mide generalmente con métricas como precisión, recall y puntuación F1, que evalúan cómo el modelo clasifica datos nuevos y desconocidos [14]. Los mismos se detallan mas adelante, en 1.4.4.5 "Métricas de evaluacion para algoritmos de ML".

1.4.4.2 Aprendizaje no supervisado aplicado al reconocimiento de imágenes de comida

El aprendizaje no supervisado, una técnica fundamental en el campo del ML. A diferencia del aprendizaje supervisado, donde se utilizan datos etiquetados, el aprendizaje no supervisado trabaja con datos no etiquetados, permitiendo a los modelos descubrir patrones y estructuras por sí mismos [16], [17]. En el aprendizaje no supervisado, los algoritmos analizan datos no etiquetados para descubrir patrones y relaciones. Esta técnica es valiosa para tareas como el agrupamiento de datos, la detección de anomalías, la reducción de dimensionalidad, entre otras [16], [17].

En el reconocimiento de imágenes, como las de comida, el aprendizaje no supervisado puede ser utilizado para segmentar objetos o regiones en una imagen. Esto es útil para identificar patrones visuales distintivos en los alimentos, ayudando a diferenciar entre di-

versos tipos de comidas[18], [19]. Algunos de los algoritmos comunes en el aprendizaje no supervisado incluyen el análisis de componentes principales (PCA), el agrupamiento k-means, y las redes neuronales autoencoder. Estos algoritmos son fundamentales para la identificación de estructuras y patrones en los datos no etiquetados [20].

A pesar de sus ventajas, el aprendizaje no supervisado enfrenta desafíos como la mayor complejidad y el costo en comparación con el aprendizaje supervisado. Además, puede ser más difícil determinar el valor y la precisión de los resultados obtenidos a través de esta técnica [21].

1.4.4.3 Preprocesamiento

El preprocesamiento de datos permite abordar problemas relacionados con la calidad y la complejidad de los datos antes de aplicar técnicas de minería de datos [22].

1.4.4.3.1 Preprocesamiento para modelos de lenguaje de aprendizaje supervisado y no supervisado [22]

En el aprendizaje supervisado, se aborda el desafío del desequilibrio de clases, donde las diferencias significativas en las probabilidades previas de las clases pueden sesgar los algoritmos hacia la clase mayoritaria. Para mitigar este sesgo, se emplean técnicas de preprocesamiento, como el resampling, que incluye métodos de undersampling y oversampling.

El preprocesamiento de datos es crucial para asegurar la calidad y eficacia de los modelos. Debido a que los modelos supervisados dependen de datos completos y libres de ruido para lograr precisión, la limpieza de datos es esencial. El preprocesamiento incluye la limpieza, normalización, transformación, integración de datos y técnicas de reducción de complejidad como se puede observar en la Figura 1.1.

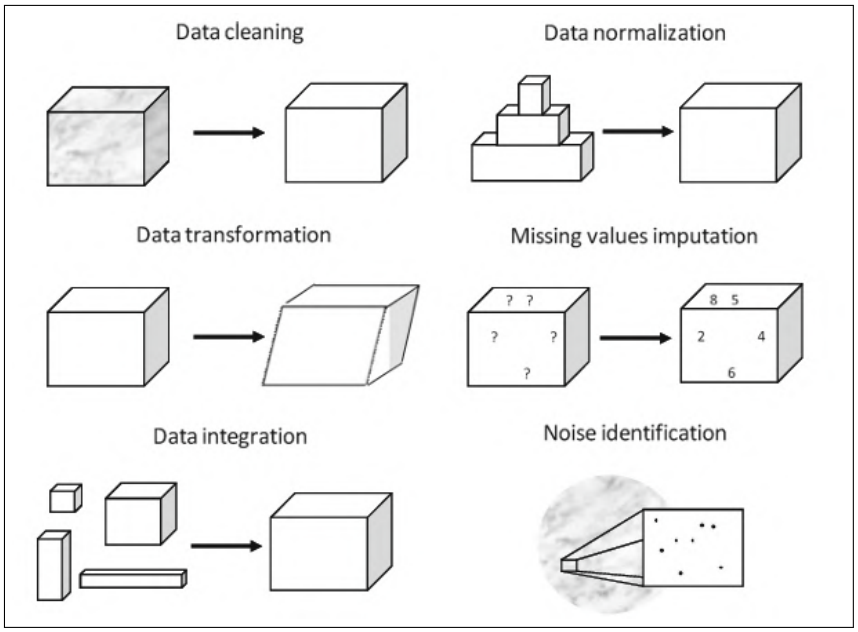

Figura 1.1: Tareas de preprocesamiento de datos [22].

En cuanto al aprendizaje no supervisado, existen enfoques similares a los del aprendizaje supervisado. El preprocesamiento se centra en técnicas de reducción de datos, selección y transformación de características. Aunque en este contexto no se necesitan etiquetas para el análisis, la corrección de ruido en los datos sigue siendo esencial. Además, la imputación de valores faltantes sigue siendo una consideración relevante para asegurar la integridad del análisis no supervisado, destacando la importancia de abordar la calidad de los datos incluso en ausencia de información de etiquetas.

1.4.4.3.2 Preprocesamiento para modelos de ML [22]

Ante el crecimiento exponencial de los datos digitales, se destaca la importancia del preprocesamiento de datos, lo que vuelve fundamental el desarrollo de técnicas y metodologías avanzadas para un procesamiento y análisis efectivos [23].

En la ciencia de datos, elegir algoritmos apropiados es fundamental. El análisis de grandes conjuntos de datos implica abordar problemas comunes categorizados en clasificación, regresión, agrupación y reducción de dimensiones. Estos problemas son cruciales para obtener información y tomar decisiones informadas [23].

El preprocesamiento en modelos de ML abarca diversas técnicas fundamentales para la gestión y preparación de datos. Existen herramientas de ML que ofrecen técnicas de preprocesamiento que se dividen en categorías como discretización y normalización, extracción de características, selección de características, indexadores y codificadores de caracterís-

ticas, y minería de texto [22].

La discretización implica la transformación de variables continuas en intervalos discretos, mientras que la normalización ajusta las características para seguir una distribución normal estándar, llevándolas a un rango específico definido por valores mínimo y máximo. La extracción de características busca reducir la redundancia y complejidad en el conjunto de datos, y la selección de características es esencial para conservar la relevancia de subconjuntos específicos sin perder información crucial. Los indexadores y codificadores son funciones que convierten características de un tipo a otro mediante técnicas específicas. Por último, la minería de texto implica aplicar técnicas para estructurar datos de texto, descubriendo patrones significativos [22]. Estas estrategias de preprocesamiento en el contexto del ML, son fundamentales para optimizar la calidad y utilidad de los datos.

1.4.4.4 Algoritmos de clustering y segmentación en el reconocimiento de imágenes de comida.

La segmentación de imágenes desempeña un papel importante en diversas aplicaciones de ML, y una de las áreas más destacadas es el reconocimiento de alimentos. En este contexto, se hace referencia a la segmentación de imágenes de comida (FIS: Food Image Segmentation), como un campo de investigación conocido. Más allá de su impacto en la visualización de imágenes, tiene relevancia al momento de estimar las calorías de los alimentos lo cual es esencial para la salud y nutrición [24]. Existen herramientas y técnicas que tienen un papel importante al momento de realizar clustering y segmentación para el reconocimiento de imágenes de alimentos. Entre estos se destacanK-Means, Random Forest y CNN con DenseNet201.

1.4.4.4.1 K-Means [25]

K-Means es una técnica de agrupamiento la cuál es utilizada en problemas de clustering como la minería de datos, el descubrimiento de conocimiento, la compresión de datos, entre otros. Esta técnica busca reducir una función objetivo formal asignando puntos de datos a un grupo de centros en un espacio d-dimensional real (R^d). Con la finalidad de identificar k centros en el espacio d-dimensional real, minimizar la distancia cuadrada media entre cada punto de datos y su centro más cercano.

Al observar que la posición más adecuada para un centro se encuentra en el centroide del clúster correspondiente, el método de K-Means, también llamado algoritmo de Lloyd, destaca como una técnica para abordar el problema de K-Means. Su funcionamiento se

apoya en la implementación de un árbol kd para estructurar datos multidimensionales.

Durante la construcción del árbol kd, se establece una estructura de árbol binario donde cada nodo está vinculado a una “celda”, la cual es un rectángulo hiperdimensional alineado con los ejes y que contiene un grupo de puntos. Inicialmente, el nodo raíz representa el rectángulo que engloba todos los datos. Después, se seleccionan k centros iniciales que se asignan a celdas específicas en el árbol kd.

Se determina el recuento de puntos y centroides ponderados en cada nodo interno del árbol. Los candidatos se propagan de forma descendiente en el árbol, seleccionando en cada nivel el candidato más cercano al punto medio de la celda. Este procedimiento se repite en iteraciones sucesivas, actualizando los centros con los centroides de los puntos asignados, hasta que el algoritmo converge y los centros dejan de experimentar cambios sustanciales.

La etapa de filtrado de candidatos se ejecuta con el propósito de determinar si un candidato puede ser excluido de la lista. En situaciones en las que ninguna región de la celda se encuentra más cercana a un candidato que a otro, es posible podar el candidato. Lo cual se logra mediante un hiperplano que divide el segmento de línea entre el candidato y su contraparte más cercana. La celda se evalúa para determinar en qué lado de este hiperplano se encuentra; si la celda se sitúa completamente en un lado, se procede a podar el candidato. El siguiente algoritmo de filtrado indica el proceso de propagación de candidatos y la asignación de puntos a los centros más cercanos.

```
1  Filter(kdNode u, CandidateSet Z){
2      C ← u.cell;
3      if (u is a leaf) {
4          z* ← the closest point in Z to u.point;
5          z*.wgtCent ← z*.wgtCent + u.point;
6          z*.count ← z*.count + 1;
7      }else {
8          z* ← the closest point in Z to C midpoint;
9          for each (z ∈ Z \ {z*})
10             if (z.isFarther(z*,C)) Z ← Z \ {z};
11
```

Código 1.1: Proceso de propagación de candidatos y la asignación de puntos a los centros más cercanos

```
        if (|Z| = 1) {
            z*.wgtCent ← z*.wgtCent + u.wgtCent;
            z*.count ← z*.count + u.count;}
        else {
            Filter(u.left, Z);
            Filter(u.right, Z);
        }
    }
}
```

Código 1.2: Continuación de Proceso de propagación de candidatos y la asignación de puntos a los centros más cercanos

Además, la Figura 1.2 muestra el concepto del hiperplano que biseca el segmento entre dos candidatos.

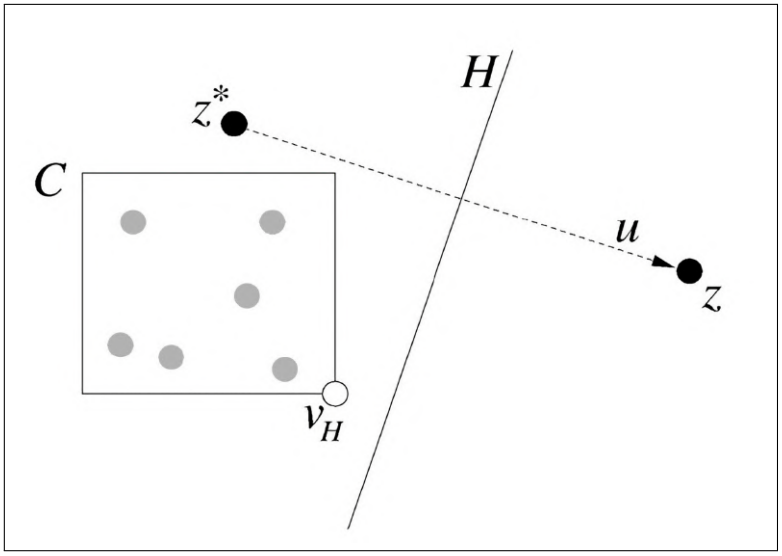

Figura 1.2: El candidato z se poda porque C se encuentra completamente en un lado del hiperplano bisector H [25].

1.4.4.4.2 Random Forest [26]

Este algoritmo, también conocido como “Bosque Aleatorio”, es un algoritmo de aprendizaje supervisado. Se utiliza tanto en problemas de clasificación como en regresión. Creando múltiples árboles de decisión durante la fase de entrenamiento, donde se selecciona aleatoriamente una muestra de datos del conjunto de entrenamiento, y para cada árbol se toma un subconjunto aleatorio de características. De tal forma, que se introduce variabilidad en el proceso de aprendizaje, algo fundamental para evitar el sobreajuste del modelo a los datos

de entrenamiento.

En la fase de predicción, cada árbol del bosque emite una predicción (en regresión) o vota por una clase (en clasificación). La predicción final del modelo se determina recopilando votos mayoritarios en los casos de clasificación o al calcular el promedio de las predicciones en situaciones de regresión como se puede observar en la Figura **??**. Al realizar esta fusión de modelos se reduce la varianza y se mejora la generalización del modelo.

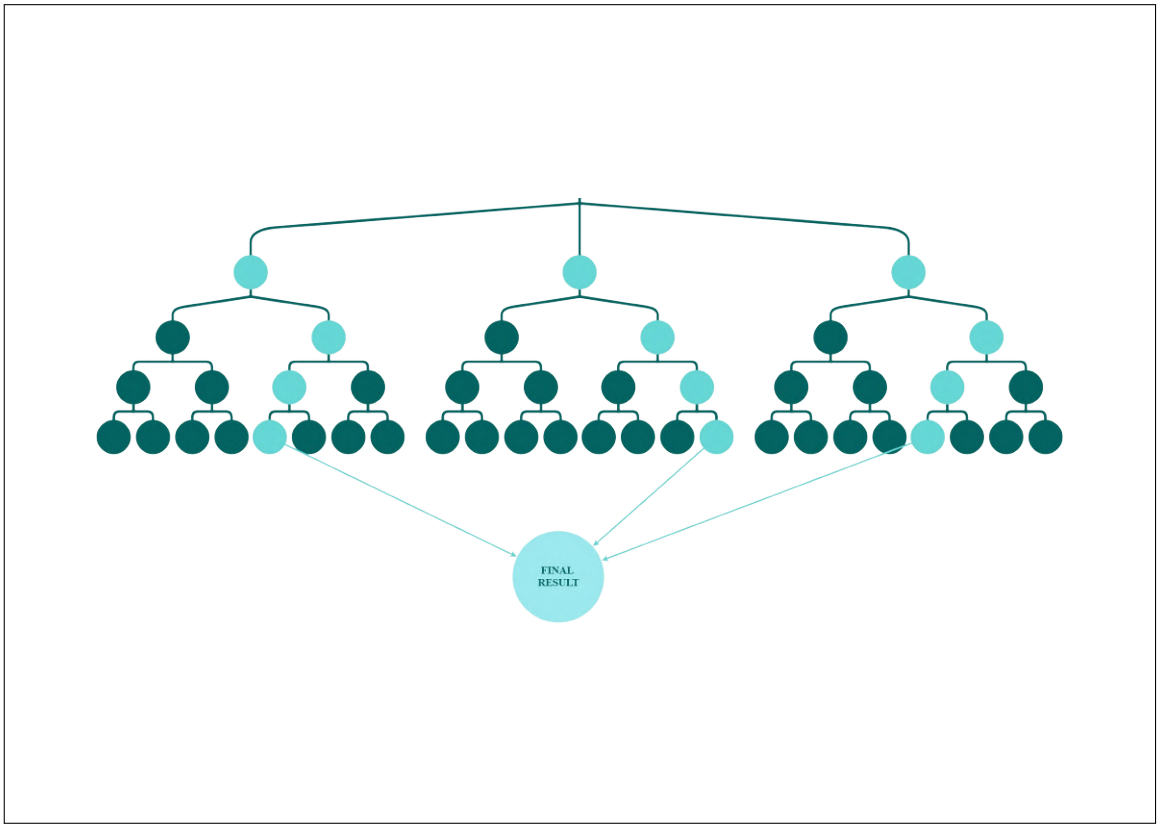

Figura 1.3: Ilustración de Random Forest

El algoritmo proporciona una medida de la importancia de las características utilizadas en el modelo. Lo cual permite evaluar las variables que tienen mayor influencia en las predicciones del modelo, proporcionando una herramienta para interpretar el modelo y elegir las características más relevantes.

1.4.4.4.3 CNN con DenseNet201

Las redes neuronales convolucionales (CNNs: Convolutional Neural Networks) son redes

neuronales que contienen neuronas con sesgos y pesos en cada capa. Un modelo CNN profundo típico incluye capas convolucionales, capas de normalización por lotes, capas de agrupación y capas totalmente conectadas o densas. La normalización por lotes no está presente en algunas arquitecturas populares de CNN, como AlexNet[4] y VGG[5], debido a que antes de estos, esta técnica no existía [27].

Las CNNs se utilizan comúnmente para procesar datos en forma de matrices, como imágenes a color bidimensionales con tres canales de color (rojo, verde y azul). Existen diferentes tipos de procesamiento de datos con las CNN. En una dimensión usadas para señales de voz, dos dimensiones para el procesamiento de datos de imágenes y tres dimensiones para el procesamiento de datos de video [28] .

El aprendizaje por transferencia es una metodología de investigación prevalente en el campo del ML. Reutiliza el conocimiento aprendido por un modelo preentrenado y entrena un nuevo modelo en un nuevo conjunto de datos. Lo cual resulta útil cuando no se dispone de una gran cantidad de datos para entrenar un modelo desde cero [27].

Existen modelos de CNN preentrenados tales como ResNet101, VGG-16, InceptionV3 y DenseNet201, utilizados para desarrollar modelos de reconocimiento de alimentos mediante transferencia de aprendizaje. Estos 3 últimos modelos fueron entrenados previamente utilizando el conjunto de datos ImageNet [27], [28].

Para le presente trabajo de integración curricular se ha utilizado el modelo DenseNet debido a un estudio en donde se revisaron diversos modelos de transferencia de aprendizaje, como ResNet101, DenseNet201 y MobileNetV2. El cual destacó que al aplicar el modelo DenseNet201 se logró una precisión del 100 % en la clasificación de imágenes térmicas de mama [28].

La arquitectura de la Red Convolutiva Densa (DenseNet) conecta cada capa directamente a la siguiente de manera feed-forward. A diferencia de las CNN tradicionales, DenseNet tiene una relación L(L+1)/2 entre capas, donde L es el número de capas. Utiliza capas estrechas con pocos mapas de características, lo que facilita el despliegue de características, fomenta la reutilización de características y reduce el número de parámetros [28] .

DenseNet-201 es una CNN de 201 capas que utiliza la base de datos ImageNet para cargar

[4] AlexNet es una red neuronal convolucional con 8 capas de profundidad.

[5] VGG es una red neuronal convolucional.

una red preentrenada con más de un millón de imágenes. La red clasifica imágenes en 1000 categorías diferentes. La entrada de imagen de la red es de tamaño 224 x 224 pixeles [28]. En cada capa, se realiza normalización por lotes, activación ReLU y convolución con un filtro. Cada bloque contiene una entrada en forma de matriz correspondiente a los píxeles de la imagen, que pasa por las etapas de normalización por lotes y activación ReLU para reducir el sobreajuste durante el entrenamiento. La convolución con un filtro de 3x3 procesa la matriz de imagen resultante para obtener una salida previamente procesada [28]. En la Figura 1.4 se puede observar la arquitectura de DenseNet.

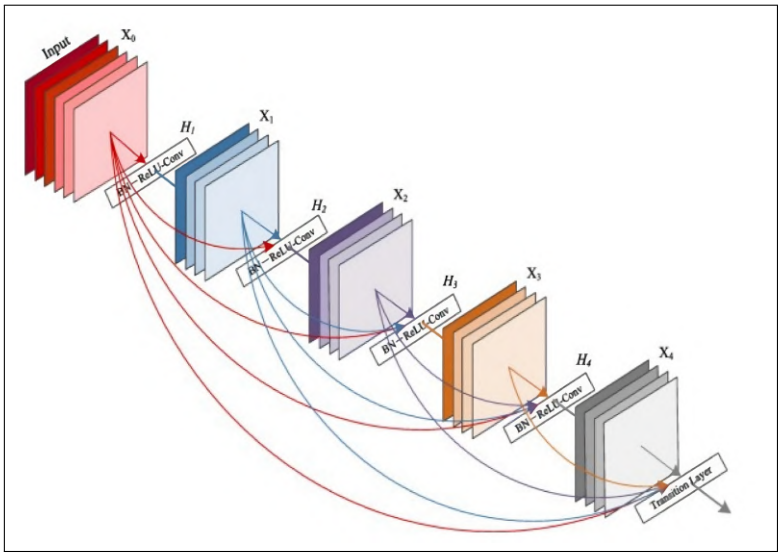

Figura 1.4: Arquitectura de DenseNet [28]

Se debe tomar en cuenta que la densa concatenación de DenseNet plantea un desafío que demanda una alta memoria de unidad de procesamiento gráfico (GPU:...) y más tiempo de entrenamiento. Por otro lado, la atenuación de las dimensiones de representación en ResNet mediante el atajo de identidad también presenta limitaciones. Existe un dilema en la elección entre estos dos modelos para muchas aplicaciones en términos de rendimiento y recursos de GPU [29].

1.4.4.5 Métricas de evaluación para algoritmos de ML [30]

El rendimiento de los modelos automáticos para el reconocimiento de alimentos se basa en la asignación adecuada de imágenes de alimentos a sus respectivas categorías. Para determinar la precisión de los modelos se lo puede realizar mediante las métricas de evaluación, para determinar la precisión de los modelos de reconocimiento de comida.

– Exactitud: Esta métrica determina si un modelo puede predecir correctamente las cla-

ses de los alimentos o su rendimiento general. Sin embargo, cuando se tiene conjuntos de datos desequilibrados, esta métrica no es del todo efectiva.

– Precisión: Se refiere a la frecuencia con la que un modelo predice correctamente valores clasificados, es decir, indica el porcentaje de clases alimentarias positivas predichas que son verdaderamente positivas.

– Sensibilidad: Se refiere a la capacidad del modelo para clasificar correctamente las clases alimentarias, es decir, indica el porcentaje de clases alimentarias positivas totales que se predice como positivas.

– Puntuación F1: Esta métrica representa la media armónica de las puntuaciones de sensibilidad y precisión. Al considerar tanto los falsos positivos como los falsos negativos. A diferencia de la exactitud, funciona bien en conjuntos de datos desequilibrados.

2 METODOLOGÍA

2.1 SELECCIÓN DEL DATASET

En la búsqueda de un conjunto de datos óptimo para el estudio en el campo del reconocimiento de imágenes de alimentos mediante técnicas de ML, se priorizaron criterios específicos que reflejan las necesidades únicas de la investigación. Siguiendo la guía "A Comprehensive Survey of Image-Based Food Recognition and Volume Estimation Methods for Dietary Assessment" [31], se puso especial atención en seleccionar un dataset que no solo fuera amplio y diverso, sino que también resonara con el contexto regional del estudio.

Uno de los principales criterios fue la relevancia regional de los alimentos incluidos en el dataset. Era esencial seleccionar un conjunto de datos que contuviera alimentos fácilmente reconocibles y comúnmente consumidos en la región del estudio [1] . Esto no solo facilitaría la identificación precisa de los alimentos, sino que también aseguraría la relevancia y aplicabilidad de los resultados en pruebas posteriores con imágenes de comida tomadas localmente. Además, el conjunto de datos ideal debería clasificarse entre los tres primeros en términos de cantidad de imágenes, ya que esto mejoraría el entrenamiento del algoritmo. [32] . Esta amplitud y profundidad son cruciales para el entrenamiento efectivo de modelos de ML y para garantizar una alta precisión [2] en el reconocimiento de imágenes. Estos criterios, junto con la necesidad de una diversidad adecuada, guiaron la selección del dataset apropiado para el estudio.

Para ilustrar más claramente la gama de datasets considerados y sus características, se adjunta la tabla de [31] en la Tabla 2.1. Esta tabla proporciona una visión detallada de los diversos conjuntos de datos evaluados, sus categorías de alimentos, y el número de imágenes, sirviendo como referencia importante para entender la base de esta selección.

[1] Se seleccionó la región de comida americana como área de estudio debido a su reconocida popularidad en Ecuador, lo que facilita la identificación de los distintos tipos de alimentos característicos de esta región.

[2] En estudios anteriores, los resultados han demostrado tasas de precisión que oscilan entre el 87 y el 92 por ciento, lo cual se considerará como indicativo de .ªlta precisión.ᵉⁿ el contexto de esta investigación [33].

Tabla 2.1: Resumen de datasets de comida [31].

Authors	Year	Dataset	Food Category	Total # Images/Class
S. Godwin	2006	Wedge Shape foods	American Foods	3 categories 1038(61)
Chen	2009	PFID	American Fast Foods	256(11)
Mariappan	2009	TADA	Artificial And Generic	5000(50)
Yanai	2010	Food-50	Japanese Foods	8500(85)
Hoashi	2010	Food-85	Japanese Foods	6512(2000)
Miyazaki	2011	Foodlog	Japanese Foods	7000
Marc Bosch	2011	FNDDS	American Foods	14,361(100)
Matsuda	2012	UECFOOD-100	Japanese Foods	192,000(208)
Chen	2012	ChineseFoodNet	Chinese dishes.	5000/50
M.-Y. Chen	2012	Chen	Chinese Foods	101,000(101)
Bossard	2014	Food-101	American Foods	100,000(101)
L. Bossard	2014	ETHZ Food-101	American Foods	25,088(256)
Kawano	2014	UECFOOD-256	Japanese Foods	1 food type
T. Stutz	2014	Rice dataset	Generic (Rice)	3583 (899)
Farinella	2014	UNCIT-FD889	Italian Foods	12625
Meyers	2015	FOOD201-Segmented	American Foods	100,000(101)
Xin Wang	2015	UPMC Food-101	Generic	2000(15)
Cioccoa	2015	UNIMB 2015	Generic	19 categories
Shaobo Fang	2015	TADA(19 foods)	American Foods	117,504(3832)
Xu	2015	Dishes	Chinese Restaurant	646(41)
Beijbom	2015	Menu-Match	Generic Restaurant	37,785(975)
Zhou	2016	Food-975	Chinese Foods	110,241(172)
J. chen	2016	Vireo-Food 172	Chinese Foods	1027(73)
Cioccoa	2016	UNIMB 2016	Italian Foods	148,408(508)
Hui Wu	2016	Food500	Generic	16,643(11)
Singla	2016	Food-11	Generic	4754(1200)
Farinella	2016	UNCIT-FD1200	Generic	808,964(43)
JaclynRichetal.	2016	Instagram 800k	Generic	2978(19)
Liang	2017	ECUSTFD	Generic	7500/15
Güngör	2017	Turkish-Foods-15	Turkish	5000(50)
Pandey	2017	Indian Food Database	Dishes	700/50
Termritthikun	2017	THFood-50	Indian Foods	247,636(524)
Ciocca	2017	FOOD524DB	Thai Foods	160,731(292)
Hou	2017	VegFru	Generic	15,630(81)
Waltner	2017	FruitVeg-81	Generic (Fruit)	71,125(103)
Muresan	2018	Generic (Fruits 360)	Generic (Fruit)	209,700(469)

Tras una revisión y comparación de los datasets, considerando los criterios previamente mencionados, se presentan los resultados resumidos en la Tabla 2.2.

Tabla 2.2: Datasets escogidos para la implementación de algoritmos

Authors	**Year**	**Dataset**	**Food Category**	Total # **Images/Class**
Bossard et al.	2014	Food-101	American Foods	100,000(101)
Meyers et al.	2015	FOOD201-Segmented	American Foods	100,000(101)
Shaobo Fang et al.	2015	TADA(19 foods)	American Foods	117,504(3832)

De acuerdo con los criterios establecidos, se decidió seleccionar el dataset Food-101 para este estudio. Esta decisión se basó en una serie de consideraciones clave que hacen que Food-101 sea particularmente adecuado para los objetivos de la investigación en el campo del reconocimiento de imágenes de alimentos. En los siguientes párrafos, se detallarán estas consideraciones, destacando algunas de sus características en la idoneidad de Food101 para nuestro estudio.

El dataset Food-101, introducido por Bossard et al. en 2014, se destaca por varias razones. En primer lugar, cuenta con una amplia gama de imágenes, ofreciendo un total de 100,000 imágenes distribuidas en 101 categorías. Esta diversidad y volumen de imágenes son ideales para el entrenamiento de modelos de ML, proporcionando una base de datos rica y variada que es esencial para mejorar la precisión y la generalización de los modelos [32].

En segundo lugar, la composición del Food-101, centrada en alimentos americanos, es coherente con el criterio de relevancia regional establecido para el estudio. Las imágenes representan una variedad de alimentos comúnmente consumidos en la región de interés.

Además, la disponibilidad del dataset Food-101 en plataformas como Kaggle facilita enormemente el acceso y la utilización de los datos. La presencia en Kaggle, una comunidad ampliamente reconocida por profesionales y entusiastas del campo de la ciencia de datos, asegura no solo la facilidad de acceso, sino también la confiabilidad y la calidad del dataset.

2.1.1 ANÁLISIS PRELIMINAR DEL DATASET

En la etapa de análisis preliminar del dataset, se procedió a la adquisición y preparación del dataset Food-101. Este proceso se llevó a cabo utilizando la plataforma Kaggle, aprovechando sus herramientas y recursos para facilitar la descarga y gestión de los datos. El primer paso consistió en instalar la API de Kaggle en el entorno de desarrollo, en este caso,

Google Colab. Posteriormente, se subió el archivo de credenciales de Kaggle (kaggle.json) para autenticar y establecer una conexión segura con la API de Kaggle. Una vez autenticados, se utilizó un comando específico para descargar el dataset Food-101 directamente desde Kaggle. Finalmente, se descomprimió el archivo zip en una carpeta designada para su almacenamiento y acceso fácil durante las fases posteriores de análisis y procesamiento de datos. Este enfoque asegura un método eficiente y reproducible para obtener el dataset, crucial para cualquier investigación basada en datos.

Con el fin de llevar a cabo el análisis y procesamiento de datos conforme a los objetivos establecidos en esta investigación, se ha optado por utilizar el entorno de Google Colab [3] . Este entorno proporciona las herramientas y recursos necesarios para ejecutar de manera eficiente los algoritmos de reconocimiento. En este contexto, se procedió a ejecutar el Código 2.1, detallado en la sección anterior, dentro de nuestro notebook en Google Colab.

```
# Instalación de la API de Kaggle e importación de API
!pip install -q kaggle
from google.colab import files
files.upload()  # Subida del archivo kaggle.json para la autenticación
# Creación de un directorio para la API de Kaggle y configuración
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
# Descarga del dataset Food-101 desde Kaggle
!kaggle datasets download -d dansbecker/food-101
# Creación de un directorio para almacenar el dataset
!mkdir train
!unzip -qo /content/food-101.zip -d train
```

Código 2.1: Configuración Inicial y Descarga del Dataset Food-101 desde Kaggle en Google Colab.

Para la gestión eficiente de las categorías de alimentos en el dataset Food-101, se imple-

[3] Colab, también conocido como Çolaboratory", te permite programar y ejecutar Python en tu navegador con las siguientes ventajas:

- ❒ No requiere configuración
- ❒ Acceso a GPUs sin coste adicional
- ❒ Permite compartir contenido fácilmente

Colab puede facilitar tu trabajo, ya seas **estudiante**, **científico de datos** o **investigador de IA**

mentó un codificador de etiquetas personalizado. Este codificador permite la conversión ágil entre las etiquetas de categorías de alimentos y sus índices correspondientes. Se seleccionaron las primeras 99 categorías del dataset, añadiendo una categoría adicional 'other' para abarcar cualquier alimento fuera de estas categorías como se indica en el Código 2.2. Este enfoque simplifica la administración de las etiquetas y facilita el análisis posterior de los datos.

```
# Lectura de las clases del dataset y selección de las primeras 99, añ
    adiendo 'other'
classes = open("/content/train/food-101/food-101/meta/classes.txt", 'r
    ').read().splitlines()
classes_21 = classes[:99] + ['other']
# Creación de una clase para codificar las etiquetas
class Label_encoder:
    def __init__(self, labels):
        self.labels = {label: idx for idx, label in enumerate(labels)}
    def get_label(self, idx):
        return list(self.labels.keys())[idx]
    def get_idx(self, label):
        return self.labels.get(label)
# Instanciación del codificador de etiquetas con las clases
    seleccionadas
encoder_21 = Label_encoder(classes_21)
# Impresión de etiquetas y sus índices correspondientes
for i in range(100):
    print(encoder_21.get_label(i), encoder_21.get_idx( encoder_21.
    get_label(i) ))
```

Código 2.2: Codificación de Clases y Etiquetas en un Conjunto de Datos de 101 Categorías de Alimentos

Se procedió a la preparación de los DataFrames para los conjuntos de datos de entrenamiento y prueba. Esta preparación implicó la lectura de los paths de las imágenes, la asignación de etiquetas correspondientes y la adición del path completo de cada imagen. Además, se ajustaron las etiquetas desconocidas o no incluidas en las categorías seleccionadas a 'other', asegurando una gestión coherente de los datos. Para esto se ejecuta el Código 2.3 y los DataFrames resultantes proporcionan una base estructurada y accesible

para el procesamiento de imágenes.

```
# Función para preparar el DataFrame a partir del path del archivo
def prep_df(path: str) -> pd.DataFrame:
    # Lectura y separación de los datos
    array = open(path, 'r').read().splitlines()

    # Construcción del path para cada imagen
    img_path = "/content/train/food-101/food-101/images/"
    full_path = [img_path + img + ".jpg" for img in array]
    imgs = [img.split('/') for img in array]

    # Conversión a array de NumPy y ajuste de etiquetas desconocidas a
    'other'
    imgs = np.array(imgs)
    for idx, img in enumerate(imgs):
        if encoder_21.get_idx(img[0]) is None:
            imgs[idx, 0] = "other"

    # Conversión del array a DataFrame y adición del path completo
    imgs = pd.DataFrame(imgs[:, 0], imgs[:,1], columns=['label'])
    imgs['path'] = full_path

    # Mezcla aleatoria de los datos en el DataFrame
    imgs = shuffle(imgs)
    return imgs

# Preparación de los DataFrames para los conjuntos de entrenamiento y
   prueba
train_imgs = prep_df('/content/train/food-101/food-101/meta/train.txt'
   )
test_imgs = prep_df('/content/train/food-101/food-101/meta/test.txt')

# Agrupación de imágenes por etiqueta para contarlas
train_imgs.groupby(["label"]).count()
```

Código 2.3: Preparación y Análisis de Conjuntos de Datos de Imágenes de Alimentos con 101 Categorías, Utilizando Codificación de Etiquetas y Mezcla Aleatoria

Para comprender mejor la distribución de las categorías de alimentos en el conjunto de datos, se realizó una visualización utilizando un gráfico de barras. Este análisis reveló la distribución y el equilibrio de clases dentro del dataset, permitiendo identificar posibles desequilibrios que podrían afectar el entrenamiento de los modelos de ML. Ejecutar el Código 2.5 permitió obtener una visualización que fue clave para planificar estrategias de balanceo de clases y para obtener una comprensión más profunda de la composición del dataset Food-101 como se observa en la Figura 2.1. Se observa que las 99 clases seleccionadas del dataset contienen la misma cantidad de imágenes.

El desbalanceo observado en la categoría 'other'se debe a que esta representa la agrupación de dos categorías distintas, las cuales se han separado con el propósito de reconocer imágenes que no pertenecen a ninguna de las 99 categorías establecidas.

```
import matplotlib.pyplot as plt

# Función modificada para visualizar la distribución de categorías usando un diagrama de pastel con leyenda en columnas y filas
def display_category_distribution_as_pie_with_multicol_legend(dataframe, categories):
    # Ajuste de etiquetas y conteo de categorías
    modified_labels = [label if label in categorias_99 else "other_category" for label in dataframe.category]
    temp_df = dataframe.copy()
    temp_df['category'] = modified_labels
    # Agrupación y conteo de imágenes por categoría
    grouped_data = temp_df.groupby("category")
    counts = [grouped_data.get_group(group).shape[0] for group in categorias_99]
    # Creación de un diagrama de pastel sin texto de porcentaje en el centro
    figure, axis = plt.subplots()
    wedges, _ = axis.pie(counts, startangle=90)
import matplotlib.pyplot as plt

```

Código 2.4: Visualización Multicolor de la Distribución de Categorías en un Conjunto de Datos de Imágenes de Alimentos usando un Diagrama de Pastel con Leyenda en Columnas y Filas

```
    # Ajustes adicionales para el gráfico
    axis.axis('equal')  # Iguala las proporciones para formar un cí
    rculo
    # Añadir leyenda con múltiples columnas y filas
    plt.legend(wedges, categorias_99, title="Categorías", loc="center
    left", bbox_to_anchor=(1, 0, 0.5, 1), ncol=3, fontsize='small')
    # Título del gráfico
    plt.title('Distribución de Categorías en el Dataset', pad=15,
    weight='bold')
    # Mostrar el gráfico
    plt.show()
# Llamar a la función con los datos de entrenamiento y las categorías
display_category_distribution_as_pie_with_multicol_legend(
    training_data, categorias_99)
```

Código 2.5: Continuación Visualización Multicolor de la Distribución de Categorías en un Conjunto de Datos de Imágenes de Alimentos usando un Diagrama de Pastel con Leyenda en Columnas y Filas

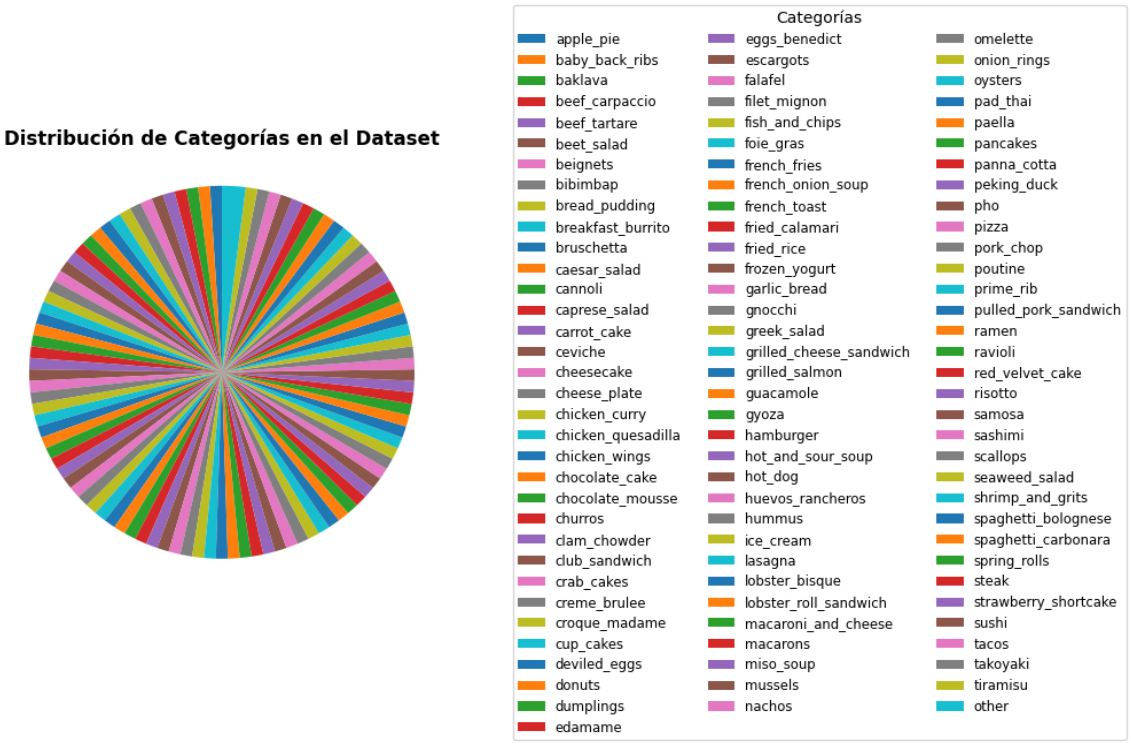

Figura 2.1: Distribución de categorías en el dataset.

Una parte integral del análisis preliminar del dataset involucró la visualización directa de las imágenes de alimentos. Para esto, se generó una muestra aleatoria de imágenes del conjunto de datos de entrenamiento, presentándolas en una matriz de subplots, lo cual se logra al ejecutar el Código 2.6. La Figura 2.2 compuesta por varias filas y columnas, ofreció una visión representativa y diversa de las categorías de alimentos presentes en el dataset. Al observar directamente las imágenes, se pudo apreciar la calidad, variedad y características específicas de los alimentos, facilitando una comprensión más profunda del contenido y la estructura del dataset Food-101. Este paso resultó crucial para evaluar la idoneidad del dataset para el entrenamiento de modelos de reconocimiento de imágenes.

```
# Configuración del tamaño del gráfico para la visualización de imá
    genes
plt.figure(figsize=(20, 5))

# Definición del número de filas y columnas para mostrar las imágenes
num_rows = 2
num_cols = 4

# Bucle para mostrar varias imágenes del conjunto de datos
for idx in range(num_rows * num_cols):
    # Selección aleatoria de un índice para elegir una imagen
    random_idx = np.random.randint(0, training_data.shape[0])
    # Lectura de la imagen seleccionada
    img = plt.imread(training_data.image_path.iloc[random_idx])
    # Obtención de la etiqueta correspondiente a la imagen
    label = training_data.category.iloc[random_idx]
    # Creación de un subplot y visualización de la imagen
    ax = plt.subplot(num_rows, num_cols, idx + 1)
    plt.imshow(img)
    plt.title(label)
    plt.axis("off")  # Ocultación de los ejes para una visualización m
    ás clara
```

Código 2.6: Visualización Aleatoria de Múltiples Imágenes del Conjunto de Datos de Alimentos en un Subplot de 2x4

Figura 2.2: Ejemplo de etiquetado de imágenes.

2.2 PREPROCESAMIENTO DE DATOS

2.2.1 PREPROCESAMIENTO PARA MODELOS DE LENGUAJE SUPERVISADO Y NO SUPERVISADO

En este apartado se aborda la preparación de imágenes para modelos de lenguaje supervisados y no supervisados. Dado el volumen del dataset original (cerca de 10 GB), se realizan ajustes para manejar eficientemente la carga de memoria y adecuar los datos a los modelos que se mencionan mas adelante. El dataset original consta de imágenes de alimentos en 101 categorías. Para adaptarlo a las limitaciones de procesamiento y memoria, particularmente para modelos de lenguaje no supervisados[4], se limita a las primeras 50 categorías. Además, se establece un máximo de 999 imágenes por categoría.

En el Código 2.8 se realiza el procesamiento en lotes con el objetivo de optimizar la memoria. Se opta por redimensionar cada imagen a 224x224 píxeles, un estándar reconocido en el campo del aprendizaje profundo que equilibra la preservación de detalles visuales importantes con la eficiencia computacional. Este tamaño se ha popularizado gracias a su uso en modelos influyentes como VGGNet [34] y se aplana[5]. Posteriormente, los datos se convierten en arrays de NumPy[6], facilitando su uso en algoritmos.

[4] Los modelos de lenguaje no supervisados requieren un enfoque diferente en el manejo de datos debido a su naturaleza y estructura de aprendizaje.

[5] El aplanamiento convierte una matriz bidimensional de una imagen en un vector unidimensional, lo que facilita su manejo como entrada para los modelos de ML.

[6] NumPy es una biblioteca para Python que soporta grandes matrices y matrices multidimensionales, junto

Finalmente, el dataset se divide en conjuntos de entrenamiento y prueba, asignando el 80 % de las muestras para entrenamiento. Esta división es crucial para una evaluación objetiva de los modelos. Este enfoque de preprocesamiento refleja una metodología eficiente y escalable, idónea para el manejo de grandes volúmenes de datos en contextos variados de ML y DL [35].

```
import os
import numpy as np
from skimage.io import imread
from skimage.transform import resize
from tqdm import tqdm

# Configuración del directorio del dataset y selección de categorías
dataset_dir = '/content/train/food-101/food-101/images'
clases = os.listdir(dataset_dir)[:50]  # Limitar a las primeras 50 categorías
batch_size = 100
longitud_deseada = 150528
train_ratio = 0.8

caracteristicas, etiquetas = [], []

# Procesamiento de imágenes en lotes para eficiencia de memoria
for i, clase in tqdm(enumerate(clases), total=len(clases), desc="Procesando clases"):
    clase_dir = os.path.join(dataset_dir, clase)
    if not os.path.isdir(clase_dir):
        continue

    imagen_nombres = os.listdir(clase_dir)[:999]  # Limitar a 999 imágenes por categoría
    for batch_start in range(0, len(imagen_nombres), batch_size):
        batch_imagenes = imagen_nombres[batch_start:batch_start + batch_size]
```

Código 2.7: Preprocesamiento y División de Conjunto de Datos de Alimentos en Características y Etiquetas para Entrenamiento y Pruebas

con una gran colección de funciones matemáticas de alto nivel para operar en estas matrices.

```
        for imagen_nombre in batch_imagenes:
            imagen_ruta = os.path.join(clase_dir, imagen_nombre)
            imagen = resize(imread(imagen_ruta), (224, 224),
    anti_aliasing=True).astype(np.float32).flatten()
            caracteristicas.append(np.resize(imagen, longitud_deseada)
    )
            etiquetas.append(i)

# Conversión a arrays de NumPy y división en conjuntos de
    entrenamiento y prueba
caracteristicas_categoria = np.array(caracteristicas, dtype=np.float32
    )
etiquetas_categoria = np.array(etiquetas)
indices = np.arange(caracteristicas_categoria.shape[0])
np.random.shuffle(indices)
limite = int(len(indices) * train_ratio)
caracteristicas_entrenamiento, etiquetas_entrenamiento =
    caracteristicas_categoria[indices[:limite]], etiquetas_categoria[
    indices[:limite]]
caracteristicas_pruebas, etiquetas_pruebas = caracteristicas_categoria
    [indices[limite:]], etiquetas_categoria[indices[limite:]]
```

Código 2.8: Continuación Preprocesamiento y División de Conjunto de Datos de Alimentos en Características y Etiquetas para Entrenamiento y Pruebas

2.2.2 PREPROCESAMIENTO PARA MODELOS DE DL

Para optimizar el rendimiento de los modelos de ML en el reconocimiento de imágenes de alimentos, se implementó una estrategia de Data Augmentation específica para el conjunto de entrenamiento. Esta estrategia incluyó la aplicación de rotación aleatoria, cambio de tamaño y recorte, y volteo horizontal de las imágenes como se puede observar en la Figura 2.3. Además, se utilizaron políticas de aumento automático basadas en las prácticas estándar para IMAGENET. La Data Augmentation es una técnica crucial para aumentar la diversidad de los datos de entrenamiento, lo que ayuda a los modelos a aprender una variedad más amplia de patrones y características, y a evitar el sobreajuste.

Figura 2.3: Ejemplo de Data Augmentation.

En contraste con las técnicas de Data Augmentation aplicadas al conjunto de entrenamiento, el conjunto de prueba se sometió a un conjunto de transformaciones más conservadoras (ver Figura 2.4). Estas transformaciones incluyeron el redimensionamiento de las imágenes a un tamaño estándar y su posterior recorte central, llevado a cabo con el uso del Código 2.9. La finalidad de estas transformaciones es mantener la consistencia y la calidad de las imágenes durante el proceso de evaluación del modelo. Al aplicar transformaciones menos variadas al conjunto de prueba, se garantiza que la evaluación del modelo se realice en condiciones controladas y estandarizadas, lo que es fundamental para obtener una medida precisa del rendimiento del modelo en datos nuevos y no vistos.

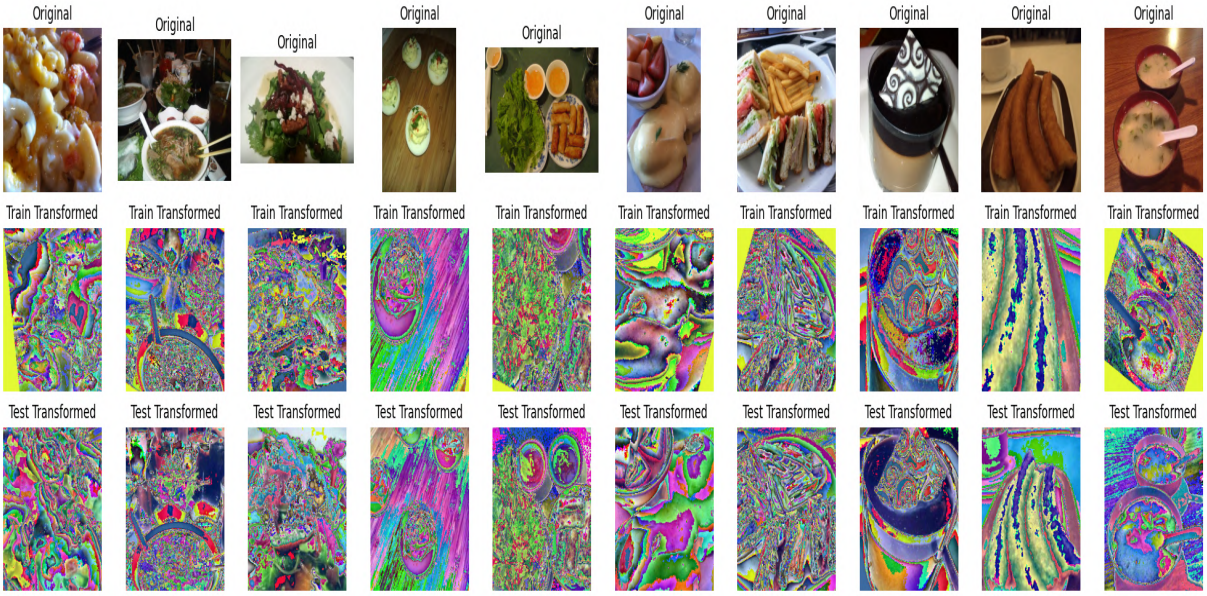

Figura 2.4: Ejemplos de imágenes transformadas.

```python
 # Implementación de aumentos de datos para el entrenamiento
train_transforms = transforms.Compose([
    transforms.RandomRotation(30),
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    torchvision.transforms.AutoAugment(torchvision.transforms.
   AutoAugmentPolicy.IMAGENET),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Implementación de aumentos de datos para las pruebas
test_transforms = transforms.Compose([
    transforms.Resize(255),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

Código 2.9: Configuración de Transformaciones para Aumento de Datos en Conjuntos de Entrenamiento y Pruebas de Imágenes

Se diseñó una clase de dataset personalizada, denominada CustomFoodDataset (Código 2.10), inherente de la clase Dataset de PyTorch. Esta clase está específicamente adaptada para manejar los conjuntos de datos de imágenes de alimentos. Su función principal es facilitar la carga y procesamiento de las imágenes y sus etiquetas correspondientes. Mediante la definición de métodos esenciales, como "***len***" para obtener el tamaño del dataset y "***getitem***" para acceder a elementos individuales, se asegura una integración fluida con las funcionalidades de PyTorch. Además, se incorporó una funcionalidad de transformación condicional para aplicar las transformaciones definidas previamente en las imágenes, permitiendo así una preparación adecuada de los datos para los procesos de entrenamiento y evaluación del modelo.

```python
from torch.utils.data import Dataset, DataLoader

class CustomFoodDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, index):
        image_path = self.df.image_path.iloc[index]
        image = Image.open(image_path)
        image = image.convert('RGB') if image.mode != 'RGB' else image
        label = encoder_21.get_idx(self.df.category.iloc[index])

        if self.transform:
            image = self.transform(image)

        return image, label

# Creación de datasets y dataloaders
train_dataset = CustomFoodDataset(training_data, transform=
    training_transforms)
test_dataset = CustomFoodDataset(testing_data, transform=
    testing_transforms)
```

Código 2.10: Definición de un Conjunto de Datos Personalizado para Imágenes de Alimentos y Creación de Datasets y Dataloaders para Entrenamiento y Pruebas

La fase final del pre-procesamiento implicó la configuración de los 'DataLoaders' para los conjuntos de datos de entrenamiento y prueba. Los 'DataLoaders' son componentes esenciales en el flujo de trabajo de PyTorch, ya que facilitan la eficiente carga y manejo de grandes volúmenes de datos en lotes durante el entrenamiento y la evaluación de los modelos. En este caso, se configuraron con un tamaño de lote de 64 y 2 trabajadores paralelos para optimizar el procesamiento. Además, el 'DataLoader' del conjunto de entrenamiento en el Código 2.11 se configuró para mezclar los datos (shuffle). Lo que asegura una variabilidad

aleatoria en cada época de entrenamiento.

Por otro lado, el 'DataLoader' de prueba no requiere mezcla, ya que se utiliza para la evaluación del modelo. Para verificar el funcionamiento correcto de los 'DataLoaders' y la integridad de los datos, se extrajeron y visualizaron las primeras cinco muestras del conjunto de entrenamiento, mostrando sus etiquetas y dimensiones, lo cual confirmó la correcta preparación y accesibilidad de los datos para las siguientes etapas del proyecto.

```
# Configuración de DataLoader para el conjunto de entrenamiento
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True,
    num_workers=2)

# Configuración de DataLoader para el conjunto de pruebas
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False,
    num_workers=2)

# Ejemplo de obtención de datos del dataset
for i in range(5):
    # Obtención de una muestra de imagen y etiqueta del conjunto de
    entrenamiento
    sample_image, sample_label = train_dataset[i]

    # Impresión de la etiqueta y el tamaño de la imagen
    print(f"Label: {encoder_21.get_label(sample_label)}, Image Size: {
    sample_image.size()}")
```

Código 2.11: Configuración de Dataloaders y Ejemplo de Obtención de Datos para Entrenamiento y Pruebas en un Conjunto de Imágenes de Alimentos

2.3 SELECCIÓN Y CONFIGURACIÓN DE MODELOS DE MACHINE LEARNING

2.3.1 IDENTIFICACIÓN DE MODELOS POTENCIALES

En el marco de la presente investigación, se contempla la integración de modelos de ML como herramientas esenciales para el desarrollo del proyecto. La selección de los modelos específicos —Random Forest [36] con asignación de etiquetas como modelo de lenguaje supervisado, el algoritmo K-Means [37] para enfoques no supervisados, y una CNN [38] como modelo de aprendizaje por refuerzo— se ha llevado a cabo cuidadosamente en función

de criterios específicos destinados a optimizar la efectividad y adecuación de los métodos empleados.

La elección de Random Fores se justifica por su probada capacidad para manejar conjuntos de datos complejos, mitigar el sobreajuste y proporcionar resultados robustos en tareas de clasificación. Al asignar etiquetas de manera supervisada, se pretende maximizar la precisión del modelo en la tarea específica de interés, lo que contribuye a la solidez de las predicciones [36].

Por otro lado, la aplicación de K-Means en el enfoque no supervisado se realiza con el propósito de explorar patrones inherentes en los datos sin depender de etiquetas preexistentes. Este algoritmo ha demostrado eficacia en la agrupación de datos, facilitando la identificación de estructuras subyacentes y relaciones entre observaciones, aspecto fundamental para la comprensión del conjunto de datos [37].

La elección de la CNN como modelo de aprendizaje por refuerzo se sustenta en su habilidad para procesar información contextualizada en imágenes, demostrando eficacia en tareas que involucran reconocimiento visual y análisis de patrones complejos [38].

En conjunto, la implementación de estos modelos se anticipa como una estrategia sinérgica, donde cada uno aportará de manera integral a la consecución de los objetivos específicos del proyecto. Se espera que esta combinación de enfoques favorezca un desarrollo sólido y exitoso de la investigación, optimizando la capacidad predictiva y descriptiva del modelo en cuestión.

2.3.2 CONFIGURACIÓN INICIAL Y ENTRENAMIENTO DE LOS MODELOS KMEANS Y RANDOM FOREST

2.3.2.1 Kmeans

En esta sección, se aborda la implementación de un modelo de ML utilizando el algoritmo K-Means. La función principal de K-Means aquí es agrupar un conjunto de datos en diferentes clusters, con el número de clusters determinado por el número de clases únicas en el conjunto de entrenamiento. Los pasos seguidos para el Código 2.12 son:

- **Determinación del Número de Clases Únicas**: Se calcula el número de clases únicas (`num_clases`) en el conjunto de entrenamiento usando la función `np.unique` de Numpy. Este paso es fundamental para establecer el número de clusters en el modelo K-Means.

- ❒ **Instanciación del Modelo K-Means**: Se crea una instancia del modelo K-Means (`kmeans`) usando la clase del mismo nombre. El número de clusters (`n_clusters`) se fija igual al número de clases únicas. Se asigna un `random_state` para asegurar la reproducibilidad.

- ❒ **Entrenamiento del Modelo**: El modelo se entrena con las características de entrenamiento (`características_entrenamiento`) utilizando `.fit()`. Este paso permite al modelo aprender a agrupar los datos en clusters.

```
# Calcular el número de clases únicas en el conjunto de entrenamiento.
num_clases = len(np.unique(etiquetas_entrenamiento))

# Crear una instancia del modelo K-Means.
# 'n_clusters=num_clases' establece el número de clusters igual al nú
    mero de clases únicas en el conjunto de entrenamiento.
kmeans = KMeans(n_clusters=num_clases, random_state=42)
print("Entrenando el modelo K-Means...")

# Entrenar el modelo K-Means con los datos de entrenamiento.
kmeans.fit(caracteristicas_entrenamiento)
```

Código 2.12: Aplicación de K-Means para Agrupar Características y Determinar el Número de Clases Únicas en un Conjunto de Datos de Entrenamiento

2.3.2.2 Random Forest

En otra sección, se implementa y entrena un modelo usando el algoritmo Random Forest. Random Forest es un método de aprendizaje en conjunto que construye múltiples árboles de decisión y produce la clase que es la moda de las clases de los árboles. El procedimiento es el siguiente:

- ❒ **Creación del Modelo Random Forest**: Se inicializa el modelo Random Forest (`modelo_rf`) con `RandomForestClassifier`, especificando el número de árboles (`n_estimators`) y un `random_state` para consistencia.

- ❒ **Entrenamiento del Modelo**: Se entrena el modelo con las características (`caracteristicas_entrenamiento`) y etiquetas (`etiquetas_entrenamiento`) usando `.fit()`. Esto permite que el modelo aprenda de los datos para realizar clasificaciones precisas en datos nuevos.

En el Código 2.13, se presenta el proceso de creación y entrenamiento de un modelo de Random Forest para clasificar datos. Se especifica un ***random_state*** fijo, lo cual es una práctica común para asegurar la consistencia y reproducibilidad de los resultados, especialmente importante en algoritmos que tienen componentes aleatorios en su ejecución, como es el caso de Random Forest. Este detalle permite que los experimentos puedan ser replicados exactamente, proporcionando una base firme para la evaluación y comparación de los resultados del modelo.

```
# Crear y entrenar el modelo Random Forest
modelo_rf = RandomForestClassifier(n_estimators=100, random_state=42)
# Obtener el número de iteraciones del ajuste
print("Entrenando el modelo Random Forest...")
modelo_rf.fit(caracteristicas_entrenamiento, etiquetas_entrenamiento)
```

Código 2.13: Creación y Entrenamiento de un Modelo Random Forest para la Clasificación de Datos

2.3.3 CONFIGURACIÓN INICIAL Y ENTRENAMIENTO DEL MODELO DE REDES CONVOLUCIONALES

2.3.3.1 Configuración Inicial de redes convolucionales

En este trabajo, utilizamos un modelo DenseNet2011 que puede visualizar en el Anexo 5.4.1 preentrenado con pesos de ImageNet3. ImageNet3 es un conjunto de datos de imágenes con más de 14 millones de imágenes y 1000 categorías. Los pesos preentrenados de ImageNet3 se pueden usar para inicializar un nuevo modelo que se va a entrenar en un conjunto de datos diferente, lo que puede mejorar el rendimiento del nuevo modelo.Este método de transferencia de aprendizaje permite un inicio más eficiente y una convergencia rápida durante el entrenamiento. Se mantendrán los pesos en las capas convolucionales para preservar la capacidad de extracción de características.

En el Código 2.14 se añade una nueva clase denominada Head, que consiste en una capa lineal con 101 unidades de entrada y 100 unidades de salida, diseñada para adaptar las dimensiones de salida a las necesidades específicas del problema. Posteriormente, el clasificador original del modelo DenseNet2011 se sustituye por esta nueva estructura.

```python
import torch
from torchvision import models
import torch.nn as nn
import requests

# Definición de la clase Head que se añadirá al final del modelo
class Head(nn.Module):
    def __init__(self):
        super(Head, self).__init__()
        self.fc = nn.Linear(101, 100)

    def forward(self, x):
        return self.fc(x)

# Inicialización del modelo con pesos preentrenados
weights = models.DenseNet201_Weights.IMAGENET1K_V1
model = models.densenet201(weights=weights)

# Congelar los pesos del modelo para evitar su actualización durante el entrenamiento
for param in model.parameters():
    param.requires_grad = False


# Combinación del modelo con la clase Head
model = nn.DataParallel(nn.Sequential(model, Head()))
```

Código 2.14: Extensión de un Modelo DenseNet201 con una Capa Lineal Personalizada para Adaptarse a un Problema Específico

2.3.3.2 Entrenamiento del modelo de redes convolucionales

Para el entrenamiento del algoritmo en el Código 2.16, se establecieron hiperparámetros [7] como el número de épocas y la tasa de aprendizaje. La función de pérdida seleccionada es

[7] Los hiperparámetros son variables que definen la estructura y cómo se entrena un algoritmo de aprendizaje automático. A diferencia de los parámetros, que son aprendidos automáticamente por el algoritmo durante el entrenamiento, los hiperparámetros se establecen antes del proceso de entrenamiento y tienen un impacto significativo en el rendimiento del modelo entrenado

la entropía cruzada[8], y se utiliza el optimizador Adam[9] por su eficiencia en el ajuste de redes neuronales.

TensorBoard[10] una herramienta relativamente nueva que promete hacer seguimiento de nuesto algoritmo se emplea para la visualización del entrenamiento y la arquitectura del modelo. La función de entrenamiento encapsula el ciclo completo, incluyendo la evaluación en un conjunto de validación y el registro de métricas, con un énfasis en la selección del modelo más óptimo basado en la precisión de validación.

```
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm

# Definición de parámetros hiperparamétricos
num_epochs = 10
lr = 1e-3

# Definición de la función de pérdida y el optimizador
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr, betas=[0.9,
    0.999])

# Configuración para visualización en TensorBoard
writer = SummaryWriter('runs/experiment_name')
sample_input_tensor = torch.randn(1, 3, 224, 224)
writer.add_graph(model, sample_input_tensor)
writer.close()
# Definición de la función para un paso de entrenamiento
def train_step(model, dataloader, loss_fn, optimizer, device):
    # Código para el paso de entrenamiento aquí "Ver condigo completo
    en Anexo "
```

Código 2.15: Entrenamiento de un Modelo de DL con Visualización en TensorBoard y Registro del Proceso

[8] La entropía cruzada es una medida de la diferencia entre dos distribuciones de probabilidad, comúnmente utilizada como función de pérdida en problemas de clasificación.

[9] Adam es un optimizador de algoritmos de ML basado en la estimación adaptativa de momentos de primer y segundo orden.

[10] TensorBoard es una herramienta de visualización para TensorFlow que permite visualizar métricas como la pérdida y la precisión, así como la arquitectura del modelo.

```python
# Definición de la función para un paso de validación/prueba
def test_step(model, dataloader, loss_fn, device):
    # Código para el paso de prueba aquí "Ver condigo completo en Anexo "

# Función para el proceso de entrenamiento completo
def train(model, train_dataloader, test_dataloader, optimizer, loss_fn, epochs, device, history=None):
    # Código para el proceso de entrenamiento "Ver condigo completo en Anexo "

# Preparación del dispositivo (GPU o CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Inicio del entrenamiento
model, history = train(model, train_loader, test_loader, optimizer, loss_fn, num_epochs, device, history=None)
```

Código 2.16: Continuación Entrenamiento de un Modelo de DL con Visualización en TensorBoard y Registro del Proceso

2.4 EVALUACIÓN Y MEJORA DE LOS MODELOS

2.4.1 EVALUACIÓN DE LOS MODELOS

2.4.1.1 K-Means y Random Forest

Como se ilustra en el Código 2.14, tras aplicar el algoritmo K-Means para la agrupación de las características de prueba, se utilizó el modelo de Random Forest para clasificar estos grupos y efectuar predicciones. Esta secuencia de pasos demuestra la integración de ambos métodos en nuestro proceso de análisis, proporcionando una estrategia comprensiva para la interpretación de los datos

```python
predicciones = kmeans.predict(caracteristicas_pruebas)
predicciones2 = modelo_rf.predict(caracteristicas_pruebas)
```

Código 2.17: Predicciones con Modelos de Agrupación y Bosques Aleatorios

2.4.1.2 CNN

En el Código 2.18 se evalúa el modelo en un conjunto de datos de prueba, calculando las predicciones y comparándolas con las etiquetas verdaderas para determinar la precisión del modelo. La utilización de `torch.no grad()` asegura que las operaciones dentro del bloque no calculen gradientes [11], lo cual es esencial durante la evaluación para reducir el consumo de memoria y acelerar el proceso.

```
def evaluate(model, dataloader, device):
    # Preparar el modelo para la evaluación
    model.eval()
    # Listas para almacenar predicciones verdaderas y predichas
    true_labels = []
    predicted_labels = []

    # Evaluar el modelo
    with torch.no_grad():
        for images, labels in tqdm(dataloader):
            # Mover datos al dispositivo (GPU o CPU)
            images = images.to(device)
            labels = labels.to(device)
            # Realizar predicciones
            outputs = model(images)
            preds = torch.argmax(torch.softmax(outputs, 1), 1)
            # Almacenar etiquetas y predicciones
            true_labels.extend(labels.cpu().numpy())
            predicted_labels.extend(preds.cpu().numpy())

evaluate(model, test_loader, device)
```

Código 2.18: Evaluación del Rendimiento del Modelo en el Conjunto de Pruebas

[11] Un gradiente representa la derivada de la función de pérdida respecto a los parámetros del modelo. Indica la dirección en la cual se deben ajustar los parámetros para minimizar la función de pérdida. La operación ***torch.no grad()*** se utiliza durante la evaluación para desactivar el cálculo de gradientes, optimizando así el uso de recursos y acelerando el proceso de evaluación, ya que la actualización de parámetros no es necesaria en esta fase.

2.4.2 RESULTADOS PRELIMINARES DE LOS MODELOS

A continuación, en la Tabla 3.2 se exponen los resultados derivados de la implementación de los modelos seleccionados. Se incluye una síntesis de los hallazgos en forma tabular. Este enfoque facilita la comparación y análisis de los datos, proporcionando una visión clara sobre las áreas susceptibles de optimización. Los hallazgos obtenidos serán fundamentales para la Sección 2.4.3, dedicada a la "Mejora de los modelos", donde se explorarán estrategias dirigidas a la mejora del rendimiento de los algoritmos en cuestión.

Tabla 2.3: Resumen Detallado de Resultados por Modelo

Modelo	Tiempo de Procesamiento	Categorías e Imágenes	Precisión
K-Means	48 horas y 11 minutos	50 categorías, 1000 imágenes por categoría	0.94 %
Random Forest	50 minutos	50 categorías, 1000 imágenes por categoría	4.65 %
CNN	1 hora y 18 minutos	100 categorías, 1000 imágenes por categoría	63.67 %

A continuación, la Tabla 2.4 especifica los recursos empleados en el proceso de entrenamiento de los modelos.Recordemos que se usa una membresía de Google Colab pro adquirida online.

Tabla 2.4: Resumen de Recursos Disponibles

Recurso	Disponibilidad
Memoria RAM Total	83.48 GB
CPU Núcleos Físicos	6
CPU Núcleos Totales	12
GPU Modelo (V100)	12 GB
Espacio en Disco Total	693.00 GB

Si bien disponemos de recursos bastante generosos comparados con PCs de uso cotidiano, es importante determinar cuántos de estos recursos se están utilizando. Esta información se presenta de forma resumida en la Tabla 2.5.

Tabla 2.5: Consumo de Recursos por Modelo.

Modelo	Memoria RAM Utilizada	Uso de CPU	Uso de GPU
K-Means	64 GB	68 %	0 %
Random Forest	48 GB	73 %	0 %
CNN	15 GB	23 %	45 %

2.4.3 MEJORA DE LOS MODELOS

2.4.3.1 K-Means y Random Forest

Como parte de la estrategia de optimización, se implementó la técnica de extracción de características utilizando un modelo preexistente de CNN. Esta aproximación permitió trans-

formar las imágenes en vectores de características, reduciendo la carga sobre la memoria al evitar el almacenamiento de las imágenes completas. La función `extract_features` se diseñó para operar en modo de evaluación y con cálculos de gradientes desactivados, lo cual minimizó el consumo de recursos y aceleró el procesamiento en el Código 2.19.

```
#Extraemos las caracteristicas directamente del modelo

def extract_features(dataloader, model, device):
    model.eval()  # Poner el modelo en modo evaluación
    features = []
    labels = []
    with torch.no_grad():  # Desactivar cálculos de gradientes para ahorro de memoria y cálculos
        for i, (inputs, label) in enumerate(dataloader):
            inputs = inputs.to(device)
            outputs = model(inputs)
            features.extend(outputs.cpu().detach().numpy())
            labels.extend(label.cpu().numpy())
    return np.array(features), np.array(labels)

# Extracción de características del conjunto de entrenamiento y de prueba
train_features, train_labels = extract_features(train_loader, model, device)
test_features, test_labels = extract_features(test_loader, model, device)
```

Código 2.19: Extracción de Características con un Modelo Preentrenado para Conjuntos de Entrenamiento y Pruebas

Para afrontar las exigencias computacionales inherentes al procesamiento de un amplio conjunto de datos, se seleccionó `MiniBatchKMeans` como alternativa a `KMeans`, buscando una mayor eficiencia en la gestión de recursos. `MiniBatchKMeans`, una variante del algoritmo `KMeans` tradicional, que se caracteriza por su capacidad para procesar subconjuntos de datos en “minilotes”, lo que reduce significativamente el tiempo de computación y el consumo de memoria.

La implementación del Código 2.20 se realizó configurando `MiniBatchKMeans` con pará-

metros específicos, tales como `init="k-means++"`, que optimiza la selección inicial de los centroides, y `n_clusters`, determinado por la cantidad de etiquetas únicas en los datos de entrenamiento, para reflejar la diversidad de categorías presentes. Otros parámetros relevantes incluyen `batch_ size=128`, que define el tamaño de los minilotes; `n_init=15`, que establece el número de inicializaciones aleatorias; y `max_no_improvement=10`, que determina el criterio de parada en ausencia de mejoras significativas.

```
from sklearn.cluster import MiniBatchKMeans

# Asegúrate de que 'train_features' y 'train_labels' sean tus datos de
    entrenamiento y etiquetas
n_clusters = len(np.unique(train_labels))  # Número de clusters basado
    en las etiquetas únicas

mbk = MiniBatchKMeans(
    init="k-means++",
    n_clusters=n_clusters,
    batch_size=128,
    n_init=15,
    max_no_improvement=10,
    verbose=0,
)

mbk.fit(train_features)
predicciones = mbk.predict(test_features)
```

Código 2.20: Clustering de Características con MiniBatchKMeans en Conjuntos de Entrenamiento y Pruebas

2.4.3.2 CNN

Para la mejora de un algoritmo de CNN, se procedió a integrar pesos preentrenados en el clasificador. Esta técnica, conocida como transferencia de aprendizaje, permite aprovechar conocimientos previos, generalmente obtenidos de un modelo entrenado en un conjunto de datos extenso, para mejorar el rendimiento en tareas específicas con menor cantidad de datos o recursos computacionales.

Para el Código 2.21, inicialmente se obtuvieron los pesos preentrenados desde una fuente externa, indicada por la URL proporcionada. Estos pesos se descargaron y almacenaron

localmente, asegurando su disponibilidad para el proceso de carga en el modelo. Posteriormente, se diseñó un nuevo clasificador compuesto por una secuencia de capas:

- Capa lineal que reduce la dimensión de 1920 a 1024, seguida de una función de activación `LeakyReLU` para introducir no linealidad.
- Capa lineal que mapea las características a 101 clases, correspondientes al número de categorías de comida en el conjunto de datos objetivo.

La configuración del modelo con el nuevo clasificador y la carga de los pesos preentrenados se realizó con atención a la compatibilidad de las dimensiones y la preservación de la arquitectura general del modelo. La utilización de `strict=False` durante la carga de pesos permitió una integración flexible, facilitando la adaptación del modelo preentrenado a la estructura específica del clasificador diseñado.

```
# Carga de pesos preentrenados para el clasificador
url = "https://github.com/Prakhar998/Food-Classification/raw/master/food_classifier.pt"
checkpoint_path = "./food_classifier.pt"
response = requests.get(url, allow_redirects=True)
with open(checkpoint_path, 'wb') as file:
    file.write(response.content)
# Configuración del nuevo clasificador para el modelo
classifier = nn.Sequential(
    nn.Linear(1920, 1024),
    nn.LeakyReLU(),
    nn.Linear(1024, 101),
)
model.classifier = classifier
model.load_state_dict(torch.load(checkpoint_path,map_location='cpu'), strict=False)
```

Código 2.21: Carga de Pesos Preentrenados para un Clasificador y Configuración en un Modelo Preexistente

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 RESULTADOS

En este capítulo, se exponen los resultados obtenidos a partir del entrenamiento de tres distintos modelos de ML. Se ha comenzado por presentar una tabla que resumirá los elementos fundamentales de cada modelo, proporcionando así una visión general y accesible de sus rendimientos. A continuación, se lleva a cabo pruebas adicionales mediante la utilización de imágenes capturadas. Estas imágenes han sidoprocesadas por el modelo que haya demostrado un mejor desempeño, con el fin de verificar su capacidad para identificar y clasificar correctamente nuevas entradas.

A continuación, se presenta una tabla detallada que especifica los recursos empleados en el proceso de entrenamiento de los modelos de ML. La tabla que se muestra a continuación detalla la cantidad de memoria RAM utilizada, el uso de la CPU y el uso de la GPU para cada uno de los modelos evaluados. Estos datos son esenciales para comprender el impacto en términos de recursos de hardware que cada modelo tuvo durante su entrenamiento.

Tabla 3.1: Consumo de Recursos por Modelo

Modelo	**Memoria RAM Utilizada**	**Uso de CPU**	**Uso de GPU**
K-Means	28 GB	68 %	0 %
Random Forest	32 GB	73 %	0 %
CNN	26 GB	23 %	39 %

Como podemos observar, el consumo de recursos se ha mantenido prácticamente constante en el caso de CNN, mientras que se ha producido una notable mejora en el caso de K-Means y RandomForest. A continuación, presentaremos los resultados obtenidos en la siguiente tabla, que resume varios aspectos importantes de los modelos analizados, para este caso se tomo 1000 imágenes por cada categoria y se usaron las 100 categorias del Dataset.

Tabla 3.2: Resumen de Resultados de los Modelos

Modelo	K-Means	Random Forest	CNN
Tiempo (min)	21	38	24
Precisión	0.0187	0.8753	0.9046
Recall	0.0187	0.8753	0.9002
F1-score	0.0187	0.8753	0.9005

Los resultados muestran una mejora significativa en el tiempo de procesamiento para K-

Means, que ha pasado de 48 horas y 11 minutos a solo 21 minutos, aunque la precisión sigue siendo muy baja. Por otro lado, Random Forest ha mejorado su precisión de 0.0465 a 0.8753 en un tiempo de procesamiento de 38 minutos. Sin embargo, la mayor mejora se observa en CNN, que ha aumentado su precisión de 0.6367 a 0.9046 en solo 24 minutos.

En resumen, estos nuevos resultados demuestran una drástica reducción en el tiempo de procesamiento y mejoras sustanciales en la precisión de los modelos, especialmente en el caso de CNN y Random Forest. Estos resultados indican un avance significativo en la eficiencia y efectividad de los modelos en comparación con los resultados anteriores.

En la Figura 3.1 se muestra un gráfico que representa la evolución de la precisión del modelo de CNN a lo largo de las épocas de entrenamiento. Este gráfico proporciona una visualización clara de cómo la precisión del modelo se ha incrementado durante el proceso de entrenamiento, lo que indica el progreso y la mejora en la capacidad de clasificación del modelo.

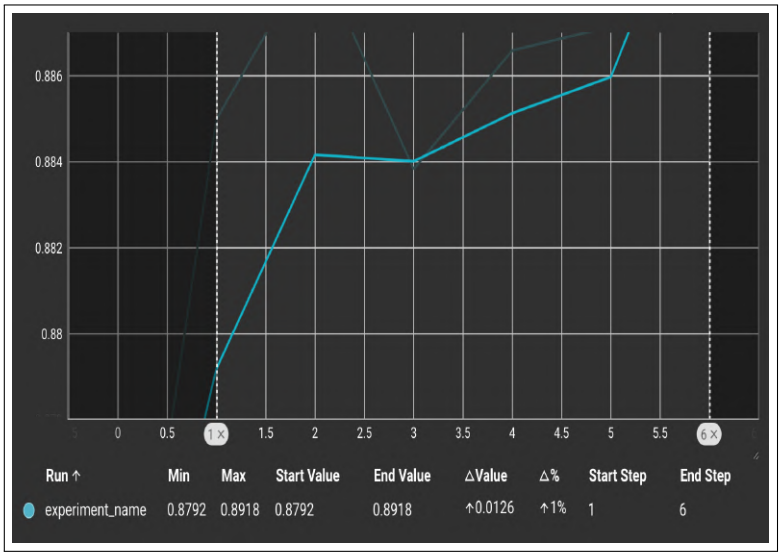

Figura 3.1: Precisión vs Épocas del entrenamiento de CNN

La Figura 3.2 presenta un gráfico que ilustra la variación de la pérdida del modelo de CNN a medida que avanza el entrenamiento. Este gráfico muestra cómo la pérdida disminuye con cada época de entrenamiento, lo que demuestra que la CNN está mejorando su capacidad para ajustarse a los datos y realizar predicciones más precisas.

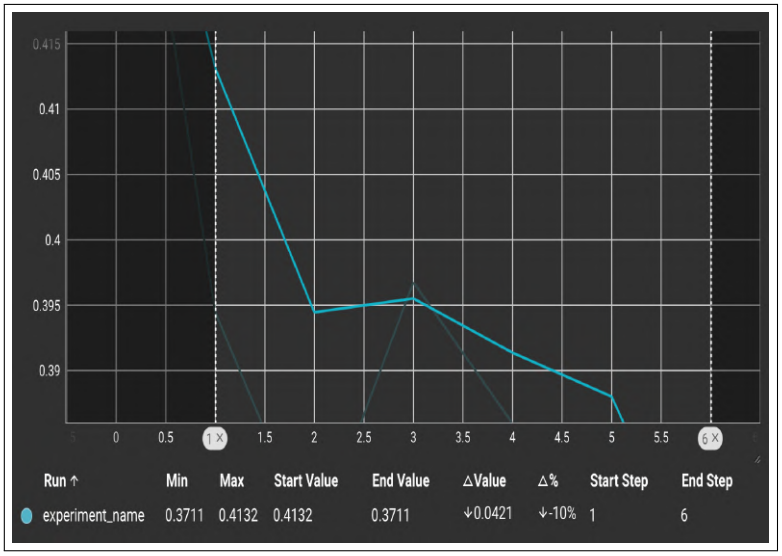

Figura 3.2: Perdida vs Épocas del entrenamiento de CNN

3.2 CONCLUSIONES

- ❒ La elección cuidadosa del dataset Food-101 se fundamentó en criterios como la relevancia regional y la diversidad de alimentos, lo cual fue crucial para la precisión y aplicabilidad de los modelos de ML en el reconocimiento de imágenes de comida. Esta selección permitió abordar el problema desde una perspectiva práctica y adaptada al contexto de interés.

- ❒ La gestión eficiente de los recursos de hardware disponibles permitió manejar grandes volúmenes de datos y realizar entrenamientos complejos, demostrando la importancia de una planificación adecuada de los recursos en investigaciones de ML.

- ❒ El empleo de técnicas de preprocesamiento y data Augmentation específicas para cada tipo de modelo optimizó la calidad y la generalidad de los datos de entrada, contribuyendo significativamente a la mejora del rendimiento de los modelos.

- ❒ La comparación de los rendimientos de estos modelos destaca la eficacia de las CNN en el reconocimiento de imágenes de comida, marcando un punto de referencia para futuras investigaciones.

- ❒ La iteración continua sobre la configuración y los parámetros de los modelos, incluida la adaptación de estrategias como la transferencia de aprendizaje para las CNN, ilustra la importancia de la mejora continua en la investigación de ML. Este enfoque adaptativo y reflexivo asegura la evolución constante de las soluciones propuestas.

3.3 RECOMENDACIONES

- Dada la intensiva demanda de recursos computacionales asociada con el algoritmo K-Means en el contexto del reconocimiento de imágenes, se recomienda investigar y aplicar métodos de agrupamiento y segmentación más eficientes. Algoritmos como DBSCAN, Mean Shift o técnicas de DL como las CNN específicas para segmentación, como U-Net o Mask R-CNN, pueden ofrecer una mayor eficiencia y precisión, siendo más adecuados para el procesamiento de imágenes en gran escala.

- Se sugiere la implementación de técnicas de optimización de recursos, como la reducción de dimensionalidad de las imágenes antes de su procesamiento o el uso de computación en la nube y GPUs para el entrenamiento de modelos. Estas estrategias pueden ayudar a mitigar los costos computacionales y hacer más accesible el análisis de grandes conjuntos de datos de imágenes.

- Realizar investigaciones que involucran el procesamiento intensivo de datos y el uso de algoritmos avanzados de ML puede resultar costoso, especialmente para estudiantes. Por lo tanto, se recomienda buscar oportunidades de financiamiento, becas o patrocinios ofrecidos por instituciones académicas, empresas del sector tecnológico o entidades gubernamentales interesadas en la investigación científica y el desarrollo tecnológico. Establecer colaboraciones con laboratorios de investigación o empresas puede proporcionar acceso a recursos computacionales y apoyo económico.

- Para minimizar costos y facilitar el acceso a datos de calidad, se aconseja el uso de conjuntos de datos públicos y la participación en proyectos de investigación colaborativa. Plataformas como Kaggle, Google Dataset Search y repositorios académicos ofrecen una amplia variedad de datasets accesibles sin costo, lo que permite realizar investigaciones relevantes sin una inversión económica significativa.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] L. Insurtech, “Así es la historia de la Visión por ordenador,” 2021. dirección: https://lisainsurtech.com/asi-es-la-historia-de-la-vision-por-ordenador/.

[2] R. Szeliski. “Computer vision algorithms and applications.” (2011), dirección: http://dx.doi.org/10.1007/978-1-84882-935-0.

[3] L. Insurtech, “Así es la historia de la Visión por ordenador,” 2021. dirección: https://lisainsurtech.com/asi-es-la-historia-de-la-vision-por-ordenador/.

[4] L. Insurtech, “Así es la historia de la Visión por ordenador,” 2021. dirección: https://lisainsurtech.com/asi-es-la-historia-de-la-vision-por-ordenador/.

[5] L. Insurtech, “Así es la historia de la Visión por ordenador,” 2021. dirección: https://lisainsurtech.com/asi-es-la-historia-de-la-vision-por-ordenador/.

[6] “Evolución de la visión artificial,” *Emergent Vision Technologies*, dirección: https://emergentvisiontec.com/es/tech-portal/evolution-of-machine-vision/.

[7] M. Velazquez, “Historia y evolución del Machine Learning,” 2018. dirección: https://recluit.com/historia-y-evolucion-del-machine-learning/.

[8] KeepCoding, “¿Qué hace un experto en visión por computadora y qué debe saber?,” 2023. dirección: https://keepcoding.io/blog/experto-en-vision-por-computadora/.

[9] DATISION, “La inteligencia artificial en el sector alimentario,” 2020. dirección: https://datision.com/blog/inteligencia-artificial-sector-alimentario/.

[10] ArticuloDeGastronomía, “MACHINE LEARNING COMO HERRAMIENTA PARA INVESTIGAR ALIMENTACIÓN Y SALUD,” 2021. dirección: https://diariodegastronomia.com/machine-learning-herramienta-investigar-alimentacion-salud/.

[11] D. Coquillat, “El sistema de reconocimiento de imágenes de Facebook que genera recetas a partir de fotografías de comida,” 2018. dirección: https://www.diegocoquillat.com/el-sistema-de-reconocimiento-de-imagenes-de-facebook-que-genera-recetas-a-partir-de-fotografias-de-comida/.

[12] D. Coquillat, “Inteligencia artificial y reconocimiento de imágenes para servir comida rápida perfecta,” 2021. dirección: https://www.diegocoquillat.com/inteligencia-artificial-y-reconocimiento-de-imagenes-para-servir-comida-rapida-perfecta/.

[13] ConectandoIdeas, “Ejemplos de Aprendizaje Supervisado: Cómo Funciona y Para Qué Sirve,” 2023. dirección: https://conectandoideas.net/aprendizaje-supervisado-ejemplos-2/.

[14] ConectandoIdeas, “Descubre los mejores ejemplos de aprendizaje supervisado,” 2023. dirección: https://conectandoideas.net/ejemplos-de-aprendizaje-supervisado/.

[15] ConectandoIdeas, “Guía completa de aprendizaje supervisado en redes neuronales: cómo funciona y cómo aplicarlo en tus proyectos,” 2023. dirección: https://conectandoideas.net/aprendizaje-supervisado-redes-neuronales/.

[16] ConectandoIdeas, “10 ejemplos de aprendizaje no supervisado que te ayudarán a entender su importancia,” 2023. dirección: https://conectandoideas.net/ejemplos-de-aprendizaje-no-supervisado/.

[17] ConectandoIdeas, “Ejemplos de Aprendizaje No Supervisado: Descubre Cómo Funciona,” 2023. dirección: https://conectandoideas.net/aprendizaje-no-supervisado-ejemplos/#google_vignette.

[18] P. Á. Corredera, “Avances de la Inteligencia Artificial: Aprendizaje No Supervisado en el Reconocimiento Visual,” 2023. dirección: https://ciberninjas.com/ia-aprendizaje-no-supervisado/.

[19] Corinna, “Aprendizaje no supervisado: Explicación del término AI,” 2023. dirección: https://www.practical-tips.com/es/internet/aprendizaje-no-supervisado-explicacion-del-termino-ai/.

[20] A. Thamm, “Aprendizaje no supervisado: Explicado de forma compacta,” 2023. dirección: https://www.alexanderthamm.com/es/blog/asi-funciona-el-aprendizaje-maquina-sin-supervision/.

[21] HistoriaDeLaEmpresa, “Aprendizaje no supervisado: Definición, beneficios y ejemplos,” dirección: https://historiadelaempresa.com/aprendizaje-no-supervisado.

[22] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez y F. Herrera, “Big data preprocessing: methods and prospects,” *Big Data Analytics*, vol. 1, n.º 1, págs. 1-22, 2016. DOI: 10.1186/s41044-016-0014-0. dirección: https://bdataanalytics.biomedcentral.com/articles/10.1186/s41044-016-0014-0.

[23] A. Rahman, “Statistics-based data preprocessing methods and machine learning algorithms for big data analysis,” *International Journal of Artificial Intelligence*, vol. 17, n.º 2, págs. 44-65, 2019. dirección: https://www.aut.upt.ro/~rprecup/IJAI_59.pdf.

[24] S. K. Abdulateef, S. R. A. Ahmed y M. D. Salman, “A novel food image segmentation based on homogeneity test of K-means clustering,” vol. 928, n.º 3, pág. 032 059, 2020. DOI: 10.1088/1757-899X/928/3/032059. dirección: https://iopscience.iop.org/article/10.1088/1757-899X/928/3/032059/meta.

[25] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman y A. Y. Wu, “An efficient k-means clustering algorithm: Analysis and implementation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, n.º 7, págs. 881-892, 2002, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2002.1017616. dirección: https://ieeexplore.ieee.org/abstract/document/1017616?casa_token=UTA-PLyCxyUAAAAA:qc2gK7lDnUs8AfI0bDqZn5ICE3a5QdcMUmu5WIfazHvsYrHgEQh45zGvImOM5rtUwMMFV-Ni9KrNGA.

[26] L. Bossard, M. Guillaumin y L. Van Gool, “Food-101–mining discriminative components with random forests,” págs. 446-461, 2014. DOI: 10.1007/978-3-319-10599-4_29. dirección: https://data.vision.ee.ethz.ch/cvl/vision2/datasets_extra/food-101/static/bossard_eccv14_food-101.pdf.

[27] A. Fakhrou, J. Kunhoth y S. Al Maadeed, “Smartphone-based food recognition system using multiple deep CNN models,” *Multimedia Tools and Applications*, vol. 80, n.º 21-23, págs. 33 011-33 032, 2021. DOI: 10.1007/s11042-021-11329-6. dirección: https://link.springer.com/content/pdf/10.1007/s11042-021-11329-6.pdf.

[28] F. D. Adhinata, D. P. Rakhmadani, M. Wibowo y A. Jayadi, “A deep learning using DenseNet201 to detect masked or non-masked face,” *JUITA*, vol. 9, n.º 1, págs. 115-121, 2021, ISSN: 2579-8901. DOI: 10.30595/juita.v9i1.9624. dirección: https://jurnalnasional.ump.ac.id/index.php/JUITA/article/view/9624.

[29] C. Zhang, P. Benz, D. M. Argaw et al., “Resnet or densenet? introducing dense shortcuts to resnet,” págs. 3550-3559, 2021. DOI: 10.1109/WACV48630.2021.00359. dirección: https://www.researchgate.net/publication/352389471_ResNet_or_DenseNet_Introducing_Dense_Shortcuts_to_ResNet.

[30] G. A. Tahir y C. K. Loo, “A comprehensive survey of image-based food recognition and volume estimation methods for dietary assessment,” vol. 9, n.º 12, pág. 1676, 2021. DOI: 10.3390/healthcare9121676. dirección: https://www.mdpi.com/2227-9032/9/12/1676.

[31] G. Tahir y L. Chu Kiong, “An Open-Ended Continual Learning for Food Recognition Using Class Incremental Extreme Learning Machines,” *IEEE Access*, vol. PP, págs. 1-1, ene. de 2020. DOI: 10.1109/ACCESS.2020.2991810.

[32] M. D. Zeiler, G. W. Taylor y R. Fergus, “Adaptive deconvolutional networks for mid and high level feature learning,” en *2011 International Conference on Computer Vision*, 2011, págs. 2018-2025. DOI: 10.1109/ICCV.2011.6126474.

[33] N. O. M. Salim, S. R. Zeebaree, M. A. M. Sadeeq, A. H. Radie, H. M. Shukur y Z. N. Rashid, “Study for Food Recognition System Using Deep Learning,” *Journal of Physics: Conference Series*, vol. 1963, n.º 1, pág. 012 014, jul. de 2021. DOI: 10.1088/1742-6596/1963/1/012014. dirección: https://dx.doi.org/10.1088/1742-6596/1963/1/012014.

[34] P. B, K. Maharajan y R. Srikanteswara, “Deep Learning Based Image Classification Using Small VGG Net Architecture,” en *2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)*, 2022, págs. 1-6. DOI: 10.1109/MysuruCon55714.2022.9972441.

[35] S. Shirmohammadi y A. Ferrero, “Camera as the instrument: the rising trend of vision based measurement,” *IEEE Instrumentation Measurement Magazine*, vol. 17, n.º 3, págs. 41-47, 2014. DOI: 10.1109/MIM.2014.6825388.

[36] Y. Liu, Y. Wang y J. Zhang, “New Machine Learning Algorithm: Random Forest,” en *Information Computing and Applications*, B. Liu, M. Ma y J. Chang, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, págs. 246-252, ISBN: 978-3-642-34062-8.

[37] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija y J. Heming, “K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data,” *Information Sciences*, vol. 622, págs. 178-210, 2023, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2022.11.139. dirección: https://www.sciencedirect.com/science/article/pii/S0020025522014633.

[38] J. Valente, J. António, C. Mora y S. Jardim, “Developments in Image Processing Using Deep Learning and Reinforcement Learning,” *Journal of Imaging*, vol. 9, n.º 10, 2023, ISSN: 2313-433X. DOI: 10.3390/jimaging9100207. dirección: https://www.mdpi.com/2313-433X/9/10/207.

5 ANEXOS

5.1 ANEXO I

5.1.1 NOTEBOOK DE GOOGLE COLAB.

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

https://colab.research.google.com/drive/1QL5lT9Pubg2DYG7bfBvTIX_aAw8BPSpV?usp=sharing

5.2 ANEXO II

5.2.1 BASE DE DATOS FOOD 101.

En el siguiente enlace se encuentra la base de datos usada en todo el proceso de entrenamiento.

https://www.kaggle.com/datasets/dansbecker/food-101/download?datasetVersionNumber=1

5.3 ANEXO III

5.3.1 MODELO BASE USADO PARA CNN.

Figura 5.1: Anexo III, Modelo base usado para CNN.

5.4 ANEXO IV

5.4.1 MODELO FINAL DEL ALGORITMO DE CNN.

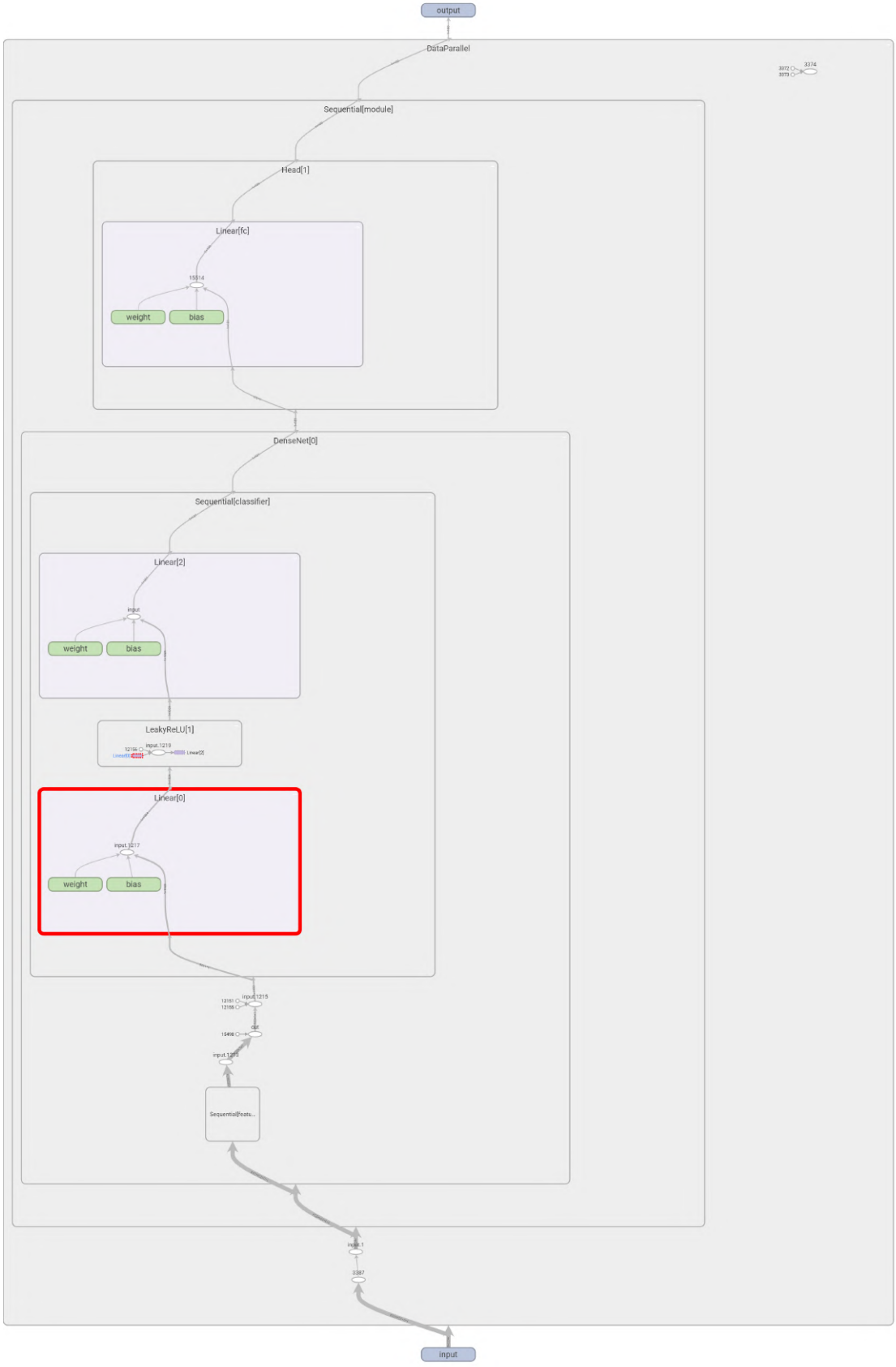

Figura 5.2: Modelo final del algoritmo de CNN.