

# **Trabajo Final Inteligencia Artificial I - Año 2023**

## **Visión Artificial y Reconocimiento de Voz**

Ingeniería en Mecatrónica

Alumno: Juan Manuel BORQUEZ PEREZ

Legajo: 13567



**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO



FACULTAD  
DE INGENIERÍA

► 1983/2023  
40 AÑOS DE DEMOCRACIA

## Contents

<b>1 Resumen</b>	<b>2</b>
<b>2 Introducción</b>	<b>2</b>
2.1 Visión Artificial . . . . .	2
2.2 Reconocimiento de Voz . . . . .	3
<b>3 Especificación del Agente</b>	<b>4</b>
3.1 Descripción y tipo de Agente . . . . .	4
3.2 Tabla REAS . . . . .	6
3.3 Descripción del Entorno . . . . .	6
<b>4 Diseño del Agente</b>	<b>7</b>
4.1 Reconocimiento de voz . . . . .	7
4.1.1 Recorte de Audios . . . . .	7
4.1.2 Extracción de Características . . . . .	9
4.1.3 Reducción de componentes . . . . .	16
4.2 Reconocimiento de Imágenes . . . . .	18
4.2.1 Separación del fondo . . . . .	18
4.2.2 Extracción de Características . . . . .	20
4.2.3 Clasificación con K-means . . . . .	21
4.2.4 Clasificación . . . . .	22
<b>5 Ejemplo de Aplicación</b>	<b>22</b>
5.1 Dataset . . . . .	22
5.2 Implementación . . . . .	22
5.2.1 Entrenamiento . . . . .	23
5.2.2 Validación . . . . .	23
5.2.3 Solución . . . . .	24
5.2.4 Ejemplo . . . . .	24
<b>6 Resultados</b>	<b>27</b>
6.1 Reconocimiento de voz . . . . .	27
6.2 Reconocimiento de Imagen . . . . .	28
<b>7 Conclusiones</b>	<b>28</b>

## 1 Resumen

En este informe se presenta el desarrollo de una solución al problema propuesto por la cátedra. Se tiene una máquina expendedora de 4 tipos de fruta: manzana, naranja, banana y pera. La máquina cuenta con una cámara para tomar fotos de las frutas en los estantes y un micrófono para solicitar frutas por voz. El software de la máquina identifica las frutas en las imágenes y sus nombres cuando son mencionadas por el usuario. La clasificación de la voz se realiza mediante un algoritmo **KNN** con  $k=3$ , y la clasificación de las imágenes se lleva a cabo con un algoritmo KNN con  $k=1$  comparando cada imagen con los centroides obtenidos del entrenamiento de un segmentador basado en **KMeans**. Se construyó un dataset disponible en línea con audios de varias personas e imágenes recopiladas en línea o tomadas por alumnos. Los resultados obtenidos fueron suficientemente satisfactorios; en concreto, la validación del modelo de reconocimiento de voz se realizó con 24 archivos de distintas personas sin falla. El reconocimiento de frutas en imágenes, aunque no completamente probado, es vulnerable ante frutas descoloridas, siendo el color la característica principal para la separación. El desarrollo se encuentra principalmente documentado en notebooks de Jupyter.

This report presents the development of a solution to the problem proposed by the department. There is a vending machine with 4 types of fruit: apple, orange, banana, and pear. The machine is equipped with a camera to take photos of the fruits on the shelves and a microphone to request fruits by voice. The machine's software identifies the fruits in the images and their names when mentioned by the user. Voice classification is done using a KNN algorithm with  $k=3$ , and image classification is performed with a KNN algorithm with  $k=1$  comparing each image with centroids obtained from training a KMeans-based segmenter. A dataset, available online, was built with audio from various people and images collected online or taken by students. The results obtained were sufficiently satisfactory; specifically, voice recognition model validation was performed with 24 files from different people without failure. Fruit recognition in images, although not fully tested, is vulnerable to discolored fruits, with color being the main feature for separation. The development is primarily documented in Jupyter notebooks.

## 2 Introducción

### 2.1 Visión Artificial

La visión artificial es la tecnología que le permite a los equipos industriales percibir las características del entorno a través de imágenes de forma automática. A diferencia de un simple procesamiento de imágenes, en el que el resultado de una imagen de entrada es otra imagen de salida modificada, la visión artificial implica la extracción de características relevantes de las imágenes que permitan identificar los elementos de interés que contienen. Las imágenes se pueden obtener con distintos tipos de sensores y es así que se tienen imágenes como las que se pueden obtener con una cámara tradicional sensible a la radiación en el rango del espectro visible o imágenes termográficas obtenidas con sensores sensibles a la radiación infrarroja del espectro por dar ejemplos.

La visión artificial clásica es un campo que se comenzó a desarrollar mucho antes del

desarrollo de las aplicaciones más avanzadas como el Machine Learning y sin embargo, a través de simples operaciones con características de las imágenes permitió identificar diferentes entidades en principio bien definidas como códigos de barras, bordes, objetos, colores, etc.

Las aplicaciones de la visión artificial son variadas e incluyen la detección de defectos en partes de máquinas, medición de partes, identificación y rastreo de objetos, identificación de textos, etc. Los principales elementos involucrados en la obtención de imágenes para la visión artificial son: una fuente de luz, un escenario específico y controlado para capturar una toma, aumentos y un sensor para capturar la imagen, en general una cámara de algún tipo.

En este trabajo se implementa la visión artificial en el sentido clásico para extraer características de imágenes de 4 tipos de frutas: peras, bananas, manzanas y naranjas con el objeto de hacer una segmentación del conjunto de imágenes en grupos según el tipo de fruta.

Inicialmente se planteó la solución al problema tratando de que sea lo suficientemente robusta como para poder identificar las frutas en cualquier tipo de fondo, en ese sentido se exploraron diversas características, máscaras y estrategias. Sin embargo, el problema de lograr la robustez no se pudo resolver de forma satisfactoria en todos los casos y por falta de tiempo se decidió tomar mayor control del escenario optándose finalmente por el uso de fondo blanco en todos los casos.

Para entrenar el segmentador fue necesario disponer de un dataset de imágenes de entrenamiento. De este dataset, algunas imágenes se recopilaron de páginas en internet mientras que la mayoría se obtuvieron tomando fotos a frutas con la cámara de un celular. En las imágenes capturadas no se tuvo demasiado recaudo en cuanto a la escena más que la utilización de luz natural y el posicionamiento de la fruta en algún fondo blanco.

Se exploraron diversas características de las imágenes como los bordes, texturas, color, etc., que fueron relevantes tanto para la separación de las frutas del fondo como para lograr la posterior segmentación del conjunto de imágenes en grupos de frutas.

## 2.2 Reconocimiento de Voz

El reconocimiento de voz es la capacidad de un sistema de software para transformar el discurso de una persona en su representación en texto, permitiendo la comunicación entre un humano y una computadora a través del habla. Este tipo de sistemas integran diferentes tipos de información contenida en la señal de audio, como la gramática, la sintaxis, la estructura y la composición del audio, incluso en presencia de ambigüedades, incertidumbres y perturbaciones como el ruido, con el objetivo de obtener una interpretación aceptable del mensaje que se desea transmitir. Estos sistemas se utilizan en aplicaciones como el dictado automático, el control por comandos de voz, traductores, reconocimiento de canciones, entre otras.

Este tipo de sistemas pueden utilizar aprendizaje deductivo o sistemas expertos, que son entrenados con los conocimientos de un conjunto de campos involucrados en el habla, tales como la lingüística, la fonética, la acústica, etc. También pueden ser sistemas que hagan uso de aprendizaje inductivo, en el cual el sistema tiene la capacidad de adquirir los conocimientos necesarios de manera automática. Dentro de esta última categoría se encuentran la mayoría de las técnicas utilizadas: Hidden Markov Models, N-Grams y



Redes Neuronales.

En el trabajo que se presenta aquí, el reconocimiento del discurso se limita a la identificación de los nombres de las frutas mencionadas. Tanto si se trata de una solución con aprendizaje automático como la solución que se presenta en este caso, en la cual no se utiliza tal técnica, es necesario llevar a cabo la extracción de las características que representan la información relevante contenida en la señal. La parte más complicada de esta solución radica precisamente en el procesamiento de las señales de audio para lograr pasar por alto perturbaciones como el silencio o el ruido, y la posterior extracción de características que permitan diferenciar audios de distintas frutas. Después de extraer un conjunto de características que permitan separar adecuadamente el conjunto de audios, la clasificación de un nuevo dato a través del algoritmo k-NN es algo trivial.

### 3 Especificación del Agente

#### 3.1 Descripción y tipo de Agente

El agente se ha interpretado de la siguiente manera. El mismo consiste en una máquina expendedora de frutas. La máquina dispone de 4 estanterías, en cada una de las cuales se encuentra uno de los tipos de fruta considerados. Cuando un usuario desea obtener una fruta de la máquina expendedora, presiona un botón para hablar en el micrófono de la máquina y decir el nombre de la fruta deseada el que el agente puede identificar a través de su programa. Luego, el agente determina si la fruta se encuentra en alguna de las estanterías y, si es así, identifica en cuál de todos. Entonces, a través de un actuador empuja la fruta del estante para expenderla al usuario. Para la determinación de la existencia y ubicación de la fruta solicitada, previo al pedido por voz del usuario, el agente toma imágenes con su cámara de las frutas en los estantes y las clasifica.

Se considera que se trata de un **agente que aprende** debido a que los algoritmos que utiliza para la clasificación de las frutas en imágenes y por voz están comprendidos dentro de ese tipo de agentes ([**RusselNorvig2009**]). El aprendizaje como tal se evidencia sobre todo en el algoritmo K-means, ya que durante el entrenamiento, el agente se vuelve capaz de encontrar similitudes y diferencias entre los grupos de imágenes. Por otro lado, en la clasificación de frutas por voz, no existe una etapa de entrenamiento como tal, y el agente requiere toda la base de datos de audio para hacer una predicción en base a una nueva orden (aprendizaje basado en memoria [6]). En ambos casos, se puede decir que el agente tiene la capacidad de mejorar su habilidad para clasificar imágenes y audio mediante la incorporación de más datos a la base de datos de imágenes y audios utilizados para el entrenamiento, otra razón por la cual se considera como un agente que aprende. En esta implementación, sin embargo, no se contempla la posibilidad de que audios de nuevas órdenes o las nuevas imágenes tomadas de las frutas en la estantería sean incorporadas a la base de entrenamiento para reentrenar al segmentador K-means o para ampliar los datos del clasificador k-NN para audio. Terminada la validación del clasificador de audios y entrenado el segmentador de imágenes, el comportamiento del agente es como el de un **agente basado en modelos** dado que son los modelos entrenados que permiten identificar los nombres y los objetos. En suma, existe durante la propia implementación del agente un proceso de ajuste no automático sino asistido por el diseñador del sistema y en base a una comparación entre el rendimiento que tiene el agente en un instante

determinado y el rendimiento esperado por el cual se considera que el agente también aprende.

### 3.2 Tabla REAS

Rendimiento	Entorno	Actuadores	Sensores
<ul style="list-style-type: none"> <li>Exactitud en el reconocimiento de las frutas medida por el número de aciertos respecto del total de ordenes del usuario.</li> <li>Rapidez en la respuesta del agente medida como el tiempo entre que el usuario lleva a cabo una orden y recibe la fruta requerida.</li> <li>Tratamiento cuidadoso de las frutas.</li> </ul>	<ul style="list-style-type: none"> <li>El gabinete de la máquina con los estantes, el estado de los mismos y la iluminación.</li> <li>El entorno en donde la máquina se ubica, su ruido ambiental y la iluminación.</li> <li>Los usuarios de la máquina y las propias frutas.</li> </ul>	<ul style="list-style-type: none"> <li>Elementos de manipulación de la cámara para desplazarla y tomar fotos en los estantes.</li> <li>Elementos para manipulación de las frutas, para colocarlas en los estantes y dispensarlas.</li> <li>Indicadores para decir al usuario si la fruta no se encuentra.</li> </ul>	<ul style="list-style-type: none"> <li>Micrófono para recibir la orden.</li> <li>Cámara para capturar imágenes.</li> <li>Botón que presiona el usuario para hacer la orden.</li> </ul>

Table 1: Tabla REAS

### 3.3 Descripción del Entorno

- Completamente observable:** Si la escena es controlada, es decir, el nivel de iluminación dentro de la cabina es suficiente, el color del fondo es el adecuado, la cámara funciona correctamente, el micrófono funciona correctamente y el ambiente no tiene demasiado nivel de ruido - como se supone - entonces los sensores permiten acceso a toda la información relevante del entorno para la toma de una decisión por parte del agente.
- Multi Agente:** Se considera que se trata de un entorno multiagente dado que la pronunciación de las frutas de una u otra forma puede tener efecto en que el agente entregue la fruta solicitada, otra diferente o ninguna, lo cual afecta el rendimiento del agente. Dicho de otra manera, el estado que percibe el agente está afectado por el comportamiento del usuario considerado en sí como un agente.
- Determinístico:** No existe fuente de elatoriedad en la operación del agente como para que la respuesta no se pueda conocer con total certidumbre. Para una misma

conjunto de imágenes de frutas y para un mismo audio de entrada, la respuesta, sea la deseada o no, siempre es la misma en distintas ejecuciones.

- **Episódico:** La clasificación del audio y de las imágenes se hace en episodios aislados. La clasificación que se haga de una próxima orden o de las imágenes de las frutas en las estanterías no depende de las clasificaciones hechas anteriormente.
- **Estático:** El agente no tiene que hacer un seguimiento del entorno mientras hace la clasificación del audio y de las imágenes dado que el mismo no cambia cuando esta haciendo una determinación; las estanterías no cambiarán hasta que se expenda una fruta y el usuario no podrá dar una orden hasta que la actual esté completa.
- **Discreto:** Como se dijo, el estado viene dado por las frutas en las estanterías y la orden del usuario. Asumiendo que el usuario, entendido como un agente, solamente solicitará frutas válidas por el micrófono, la cantidad de posibles órdenes en un instante determinado son solamente 4 (pera, banana, manzana o naranja). De la misma manera, en 4 estanterías, cada una de las cuales alberga una fruta de 4 tipos diferentes, la cantidad de posibles combinaciones será de 256. En total habrá solamente 1024 posibles, es decir, la cantidad de estados es contable y finita.

## 4 Diseño del Agente

Para el diseño de ambos sistemas se realizaron variadas pruebas que son muy extensas como para documentar en este informe, por lo que se decidió presentar aquí solamente una descripción del diseño final de los sistemas con algunas descripciones de la evolución y justificaciones de diseño. Sin embargo, está disponible en línea en un repositorio de GitHub [3] toda la investigación realizada junto con los datasets que se utilizaron.

### 4.1 Reconocimiento de voz

La principal fuente de información que se utilizó fue una lista de videos [5], acompañada de un repositorio en GitHub [4] centrado en la extracción de características para el reconocimiento de voz y música.

#### 4.1.1 Recorte de Audios

Uno de los principales problemas que se tuvo que resolver fue el de recortar los audios para preservar únicamente la parte hablada de los mismos. Inicialmente, esto se llevó a cabo con funciones de librerías cargadas, como `librosa.trim`, para la que hay que definir un umbral por debajo del cual algo es considerado como silencio. Este tipo de solución no pareció ser tan robusta, sobre todo cuando los audios presentaban cierto nivel de ruido tanto al inicio como al final del audio, dado que superaban el nivel considerado como silencio. Luego de la exploración de diversas alternativas propias, se concluyó con una solución suficientemente robusta para el recorte de los audios.

Se pudo observar que el **flujo espectral** era una excelente característica para identificar las partes habladas de un audio de las partes no habladas, siendo poco sensible a los ruidos. El flujo espectral es una característica del audio muy útil en la identificación

de eventos de sonido. Se calcula a partir de un spectrograma de magnitudes (energía) calculando la diferencia entre frames sucesivos, esto se eleva al cuadrado para eliminar el efecto de pequeñas variaciones y se suma a lo largo de todos los intervalos de frecuencia para obtener un valor para cada frame. En la Figura 1 se muestra un ejemplo de cómo el flujo espectral indica el comienzo y finalización de una parte hablada. En la misma, para un audio de ejemplo en el que se menciona la fruta naranja se superponen la señal original y el flujo espectral normalizados en el rango de -1 a 1.

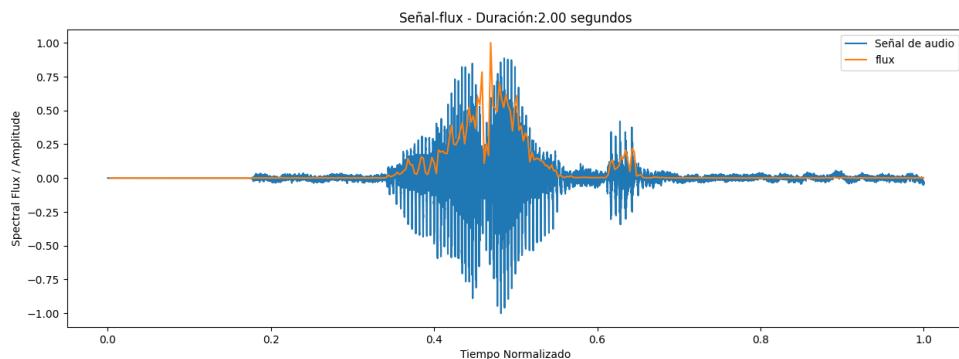


Figure 1: Flujo Espectral sobre la señal de Audio

Para producir el recorte del audio con esta propiedad basta con definir un umbral y proceder. Sin embargo, este corte es sensible a ciertas perturbaciones en el audio que se presentan como picos iniciales y finales en la señal. En ese caso, hay que definir un umbral suficientemente grande de modo de pasar por alto esos picos. Al hacer eso, el audio queda recortado de más, eliminando partes del audio habladas en los extremos, como sucedería en el ejemplo que se muestra en la Figura 2.

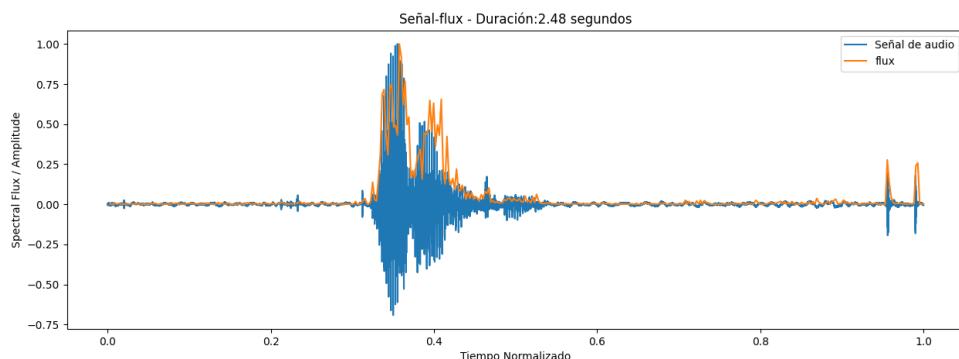


Figure 2: Flujo Espectral - Audios con Picos

Como estrategia para resolver este problema, se propone definir un umbral mínimo y un umbral máximo. El umbral máximo debe ser tal que permita pasar por alto los picos, y luego el umbral mínimo sirve como ajuste fino del corte. De esa manera, se buscaría en la señal de flujo espectral el primer instante a la izquierda y a la derecha del audio en donde se supere el umbral máximo, y desde ese punto y buscando hacia la izquierda en el extremo izquierdo o hacia la derecha en el extremo derecho encontrar el primer instante de tiempo en el que la señal de flujo espectral se encuentre por debajo del umbral mínimo.

El problema que se presenta en este caso es que existen audios en los que el flujo espectral es prácticamente nulo aún en partes habladas, como en el ejemplo de la Figura 3. Esto hace que el umbral mínimo deba ser prácticamente igual a cero.

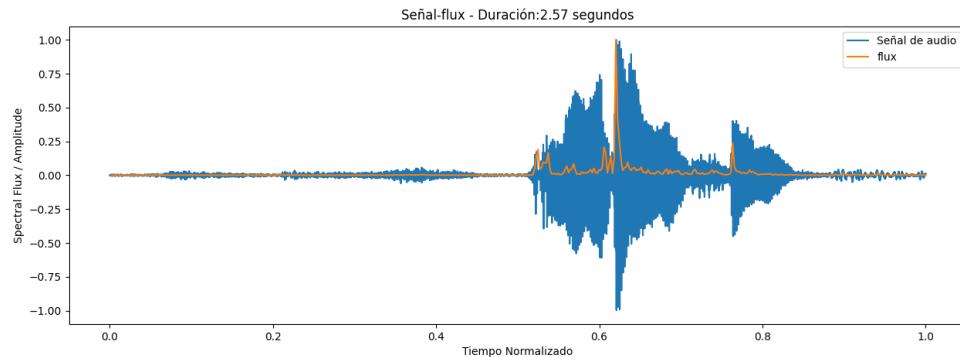


Figure 3: Flujo Espectral - Umbral mínimo

Para resolver este problema, se introduce una segunda característica el valor **RMS** de la señal que sirve como envolvente de la señal original. Ahora, la estrategia es la misma, pero el umbral mínimo se define a partir de una fracción del valor RMS y no a partir de una fracción del valor del flujo espectral. En la Figura 4 se muestra un ejemplo de este corte. En esa figura, la línea horizontal de color rojo indica el umbral de corte grueso por flujo espectral, mientras que la línea horizontal de color azul indica el umbral de corte fino por RMS. Las líneas punteadas verticales indican los puntos de corte, las rojas indican los puntos determinados por el corte grueso, mientras que las azules indican los puntos finales de corte fino por RMS. Como se puede observar, ahora es posible superar los picos finales que se presentan en la señal de audio.

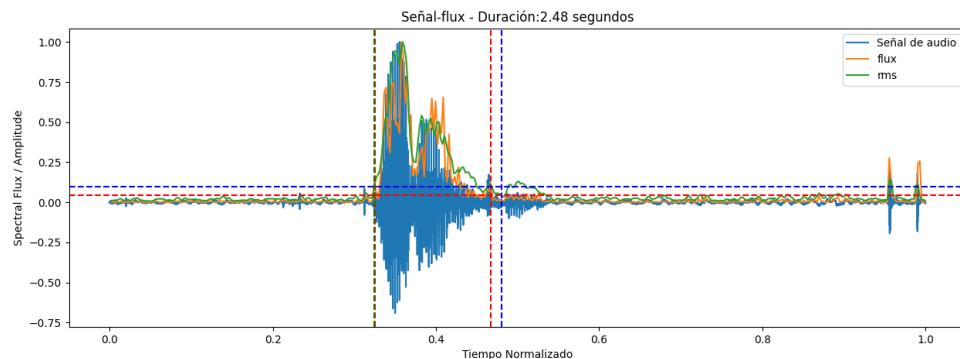


Figure 4: Flujo Espectral - RMS - Ejemplo de Corte

#### 4.1.2 Extracción de Características

En las figuras de esta sección, en el eje  $X$  se representa la razón entre el valor RMS de las señales de audio y el valor máximo de la señal, mientras que en el eje  $Y$  se representa la característica de que se trate. Los colores de los puntos se corresponden con los colores de las frutas que representan.

La extracción de características comenzó con pruebas con los coeficientes de Mel, **MFCC** (Mel Frequency Cepstral Coefficients por sus siglas en inglés), dado que la investigación arrojó que los mismos son características ampliamente utilizadas en el reconocimiento de voz. Estos tienen la capacidad de describir los fonemas (unidades de sonido de un idioma) y toman en cuenta la percepción del oído humano al utilizar la escala logarítmica de Mel para la representación de características en función de la frecuencia.

En primer lugar, se probó utilizando la media de cada coeficiente de Mel a lo largo de la duración del audio, conservando aquellas componentes que producían la mayor contribución a la separación o que tenían la menor variación dentro de cada grupo. Varias otras pruebas se realizaron con los coeficientes de Mel, por nombrar otra, se probó la utilización de los valores de los mismos a lo largo de todo el audio dispuestos en un solo vector largo, para lo que se tuvo primero que normalizar los audios en amplitud y en duración sin lograr tampoco una separación y agrupamiento satisfactorio.

En el camino, se descubrió una técnica denominada Análisis de Componentes Principales, **PCA** (Principal Component Analysis, por sus siglas en inglés), que permite la reducción de un conjunto de  $k$  observaciones en un espacio  $m$ -dimensional a un conjunto de  $k$  observaciones en un espacio  $n$ -dimensional con  $n < m$ , conservando la mayor cantidad posible de variación a través de los datos, pero de modo tal que las componentes del nuevo espacio son linealmente independientes entre sí.

Al no obtener los resultados esperados haciendo uso solo de los MFCCs, es que se decidió hacer pruebas con otras medidas agregadas del audio, entre ellas **BER** (Band Energy Ratio, por sus siglas en inglés), **ZCR** (Zero Crossing Rate, por sus siglas en inglés), la envolvente del audio, etc.

A continuación, se detallan aquellas características que finalmente se utilizaron.

- **BER:** Esta medida proporciona información sobre cómo está distribuida la energía en distintas partes del espectro de frecuencia. En esta solución se calcula como la fracción de la energía comprendida por debajo de cierta frecuencia de corte.
  - **Máximo:** Se utiliza el máximo del BER para una frecuencia de corte de 600 Hz. En la Figura 5 se muestra cómo se puede lograr una separación de las peras respecto de los demás.
  - **Mínimo:** Se utiliza el mínimo del BER a las frecuencias de corte de 1900, 5000 y 9000 Hz, en las Figuras 6, 7 y 8, respectivamente. Como se puede ver, la primera permite la separación de las peras respecto de los demás, la segunda logra una separación de las bananas respecto de las manzanas y la última una separación de las manzanas respecto de los demás, observándose cierta estratificación de los grupos en el medio.
  - **Desviación estándar:** Para el BER normalizado y considerado respecto de la media. Se tomó con frecuencias de corte a 8000 Hz (Figura 9) y 1000 Hz (Figura 10). Se puede observar cómo en el primer caso se logra una separación de las manzanas respecto de los otros grupos mientras que en el segundo caso se logra una separación de las peras respecto de los otros grupos.
- **Zero Crossing Rate (ZCR):** Esta medida cuenta la cantidad de veces que una señal cruza el eje horizontal (cero) en un intervalo de tiempo dado. El ZCR expresa

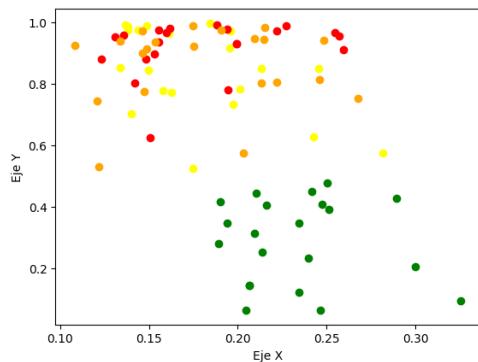


Figure 5: Máximo BER

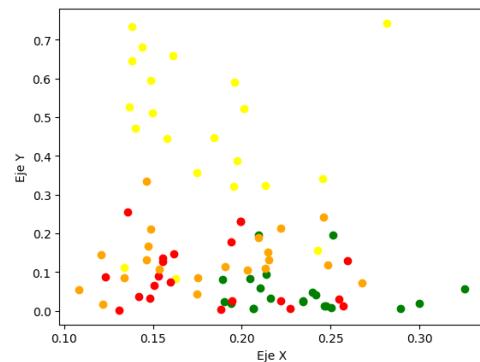


Figure 6: Mínimo BER - 1900 Hz

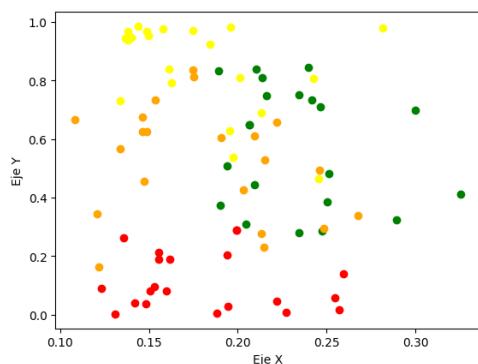


Figure 7: Mínimo BER - 5000 Hz

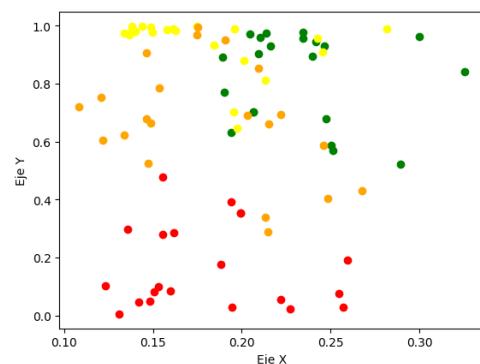


Figure 8: Mínimo BER - 9000 Hz

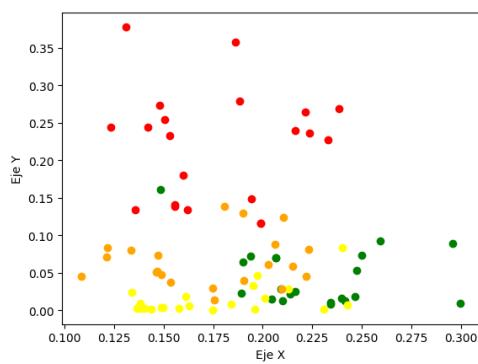


Figure 9: Std BER - 8000 Hz

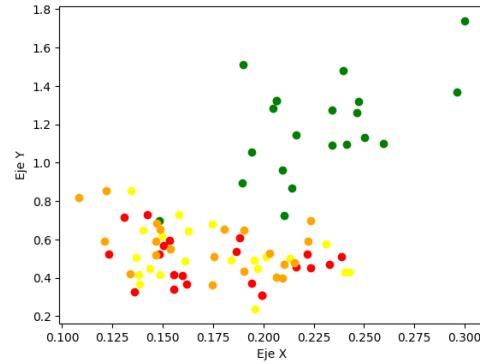


Figure 10: Std BER - 1000 Hz

una tasa, representando la frecuencia con la que la señal cambia de polaridad. Un ZCR alto indica que la señal cambia de polaridad con frecuencia. Por otro lado, un ZCR bajo indica que la señal mantiene la misma polaridad durante un período de tiempo prolongado, lo que podría ser característico de señales más suaves.

- **Media:** Se obtiene respecto del valor máximo luego de un filtro pasa banda con corte en 1000 y 5000 Hz. Como se puede ver en la Figure 11 esto logra la separación de las manzanas respecto de los demás grupos. Eso se debe a una variación de esta propiedad que no presentan el resto de los grupos cuando se pronuncia la letra 'z'.

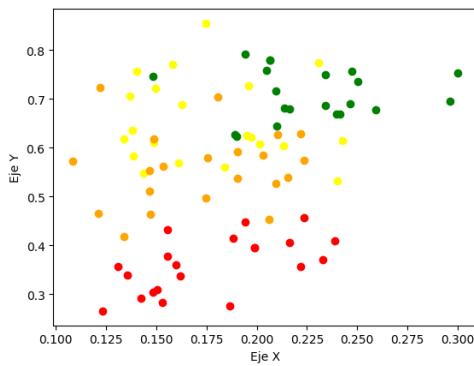


Figure 11: Zero Crossing Rate - Media

- **Máximo:** Se obtiene luego de un filtro pasa banda con cortes en 10 y 1000 Hz. En la Figura 12 se puede observar como nuevamente las manzanas se separan del resto de los grupos quedando las peras y las manzanas en grupos separados.

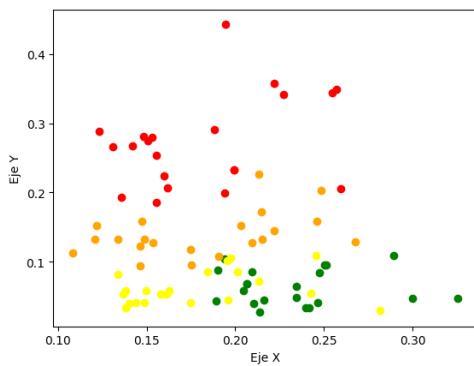


Figure 12: Zero Crossing Rate - Máximo

- **Desviación Estándar:** Se obtiene respecto de la media luego de un filtro pasa banda con cortes en 20 y 10000 Hz. En la figura 13 se puede ver cómo las manzanas se separan del resto y en el centro se pueden observar un grupo casi solo de naranjas.
- **Media a 3/14:** Luego de un filtro pasabanda con cortes en 1000 y 5000 Hz se calcula la media del audio normalizado en ese punto en la duración del audio

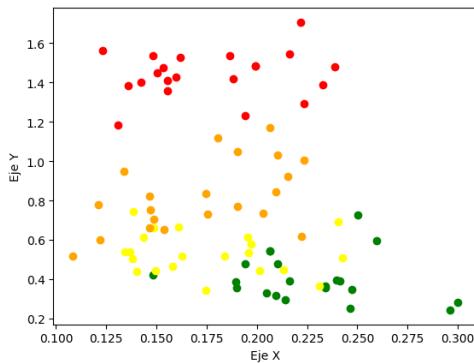


Figure 13: Zero Crossing Rate - Desviación Estándar

a lo largo de 10 frames, 5 a cada lado. Nuevamente se observa una separación de las manzanas (Figure 14)

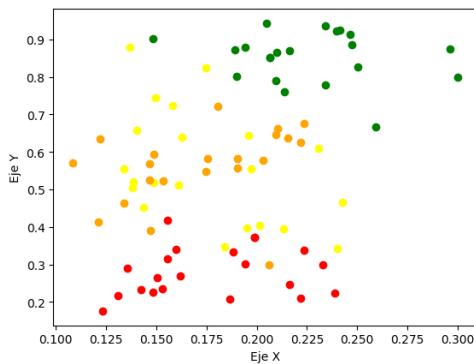


Figure 14: Zero Crossing Rate - Media a 3/14

- **Máximo a 3/4:** Luego de un filtro pasabanda con cortes en 10 y 10000 Hz se calcula el máximo en ese punto en la duración del audio a lo largo de 20 frames, 10 a cada lado buscando resaltar las diferencias entre las naranjas y las demás frutas cuando se pronuncia la letra 'j' (Figura 14).

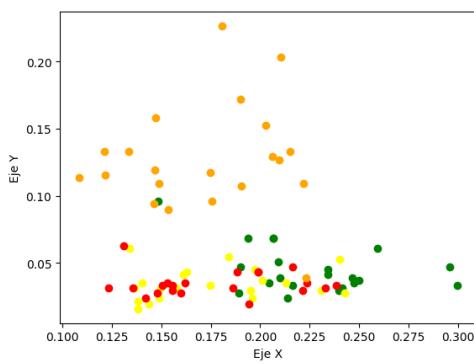


Figure 15: Zero Crossing Rate - Máximo a 3/4

- **Spectral Roll Off:** El *Spectral Roll-off* es una medida que indica la frecuencia por debajo de la cual se encuentra un cierto porcentaje de la energía total del espectro. Un valor bajo indica una concentración en frecuencias bajas, mientras que un valor alto implica una distribución hacia frecuencias más altas.
  - **Media:** Se obtiene respecto del máximo luego de un filtro pasa banda con cortes en 100 y 8500 considerando un porcentaje del 28% de la energía del espectro. Se separan las manzanas y las peras también las naranjas de las bananas y de las peras aunque queda un solapamiento entre bananas y naranjas (Figura 16).

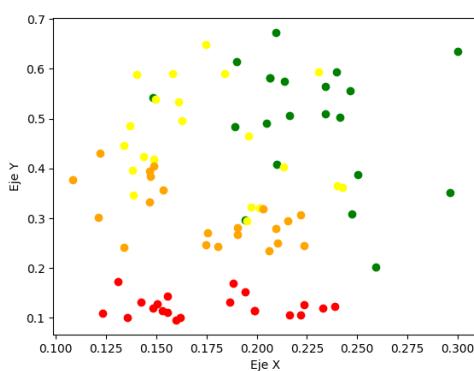


Figure 16: Spectral Roll Off - Media

- **Máximo:** Se obtiene luego de un filtro pasa banda con cortes en 100 y 8500 considerando un porcentaje del 55% de la energía del espectro. Se observa una separación de las manzanas (Figura 17).

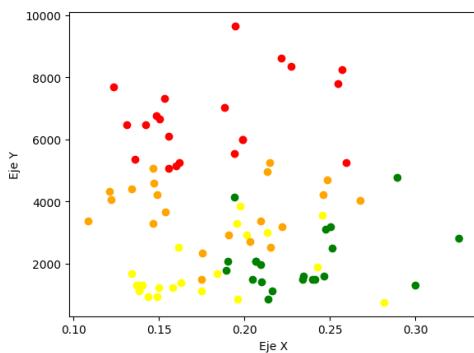


Figure 17: Spectral Roll Off - Máximo

- **Desviación Estándar:** Se obtiene respecto de la media luego de un filtro pasa banda con cortes en 50 y 8500 considerando un porcentaje del 28% de la energía del espectro. Nuevamente se observa una separación de las manzanas (Figura 18).
- **MFCCs:** Como ya se mencionó, es de las principales características utilizadas para el reconocimiento de voz dado que tiene la capacidad de identificar fonemas y demás.

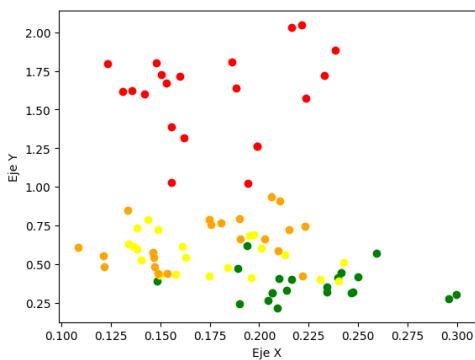


Figure 18: Spectral Roll Off - Desviación Estándar

- **Máximo del coeficiente 3:** Se obtiene luego de un filtro pasa banda con cortes en 500 y 5000. Se separan las manzanas (Figura 19).

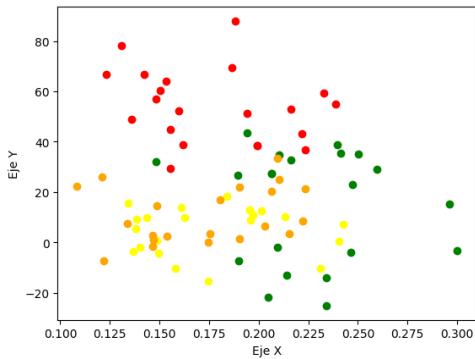


Figure 19: MFCCs - Máximo del coeficiente 3

- **Desviación estándar coeficiente 1 a 4/5:** Se obtiene respecto de la media luego de un filtro pasa banda con cortes en 10 y 8000 en 20 frames alrededor de este punto en la duración del audio (10 frames de cada lado). Aunque existe cierta dispersión, se observa una separación de las naranjas (Figura 20).

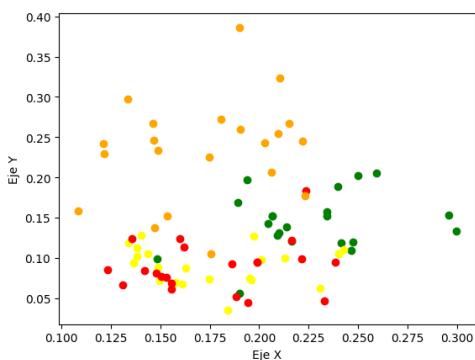


Figure 20: MFCCs - Desviación estándar coeficiente 1 a 4/5

- **Envolvente:** El cálculo del valor RMS en cada frame en que se divide un audio funciona como una buena envolvente de amplitud de la señal. De esta envolvente se toman 30 componentes equi-espaciadas de las cuales se conservan la componente 11 y la 12 (Figura 21). Dentro de todo se logra una separación de las manzanas respecto al resto

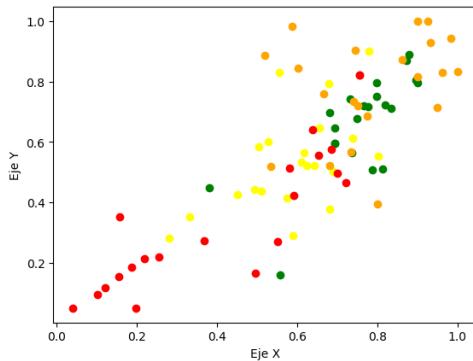


Figure 21: Envolvente componentes 11 y 12

Como se puede notar, algunas de las características extraídas son demasiado específicas. Sin embargo, este conjunto de características fue seleccionado después de muchas pruebas, al considerarse que lograban los mejores resultados en términos de agrupamiento y separación.

#### 4.1.3 Reducción de componentes

La cantidad de componentes del vector de características de cada audio es de 17. Se pretende hacer la visualización en un gráfico de los agrupamientos que se consiguen, y para eso, luego de la extracción de las características de los audios se utiliza PCA para reducir el conjunto de componentes a 3. En la Figuras 22 y 23 se observa el agrupamiento conseguido.

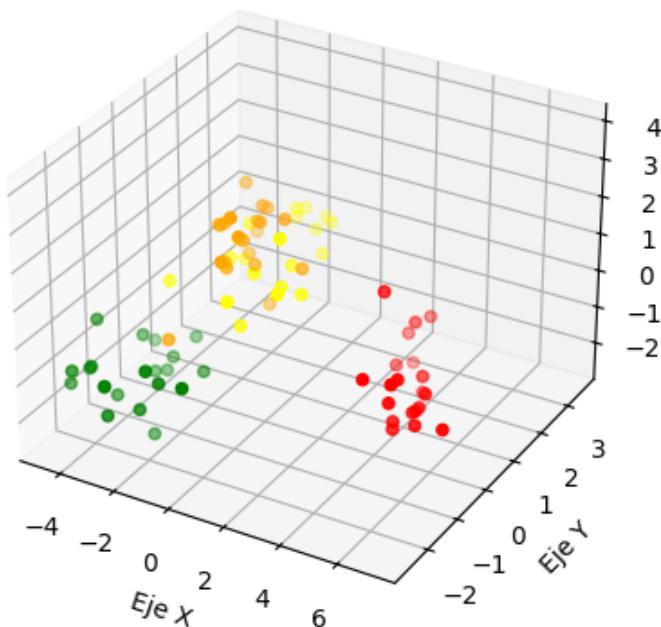


Figure 22: Agrupamiento post PCA - 1

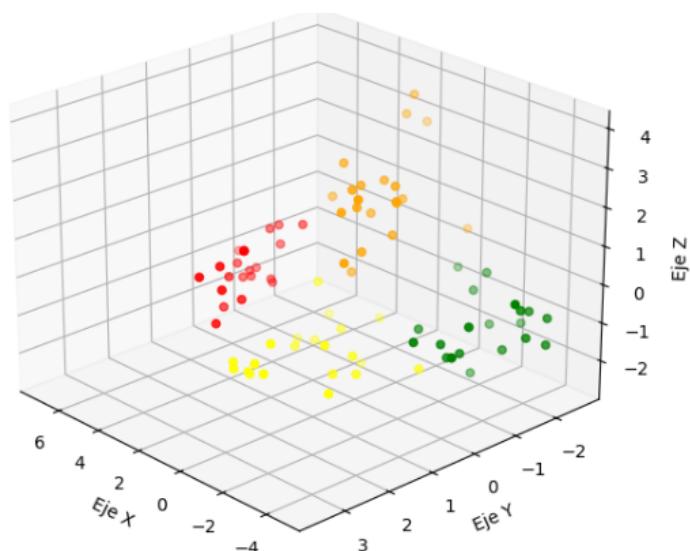


Figure 23: Agrupamiento post PCA - 2

## 4.2 Reconocimiento de Imágenes

### 4.2.1 Separación del fondo

Para lograr la separación de la fruta del fondo se exploraron diversas alternativas. A continuación, se describe brevemente el proceso indicando los resultados o los pasos de las partes que quedaron en la solución final.

Principalmente, el problema se abordó tratando de que las imágenes pudieran ser tomadas en diferentes tipos de fondos y no en uno solo normalizado, para que, por ejemplo, sea posible sacar la foto de la fruta sobre una mesa de cualquier tipo. Siguiendo esta línea, lo primero que se exploró fue la detección de contornos utilizando filtros de Sobel (figure 24), sin embargo, esto también detectaba las diferentes texturas que podía llegar a tener el fondo, con lo cual no era una solución viable, ya que no se encontraba forma de separar el contorno de la fruta del resto de los contornos. Entonces, se exploró el uso de máscaras

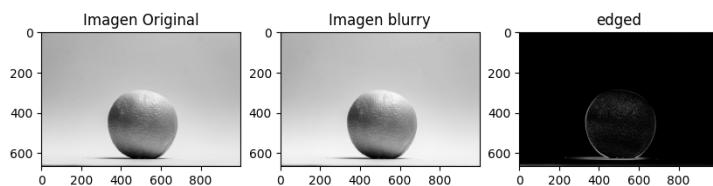


Figure 24: Ejemplo de aplicación de filtros de Sobel

de color que pudieran detectar cualquiera de los colores de las frutas que se consideran (naranja, verde, amarillo y rojo). Estas máscaras se diseñan en el espacio de colores HSV (Hue, Saturation, Value por sus siglas en inglés), que es una escala cómoda para el manejo de colores, dado que distribuye distintos colores en un cilindro en el que el color varía según el ángulo, la iluminación en la altura y la saturación o pureza del color en el radio, permitiendo pasar de forma continua por todo el rango de lo que se puede llegar a considerar rojo, por poner un ejemplo. El inconveniente se presentaba en estos casos cuando el fondo tenía colores parecidos a los mencionados, dado que quedaban incluidos dentro de lo que se consideraba como fruta.

Por este motivo, se comenzó un trabajo de pruebas de distintas máscaras para evaluar el comportamiento de las mismas. Las máscaras se construían considerando canales de distintas representaciones de la imagen. Por ejemplo, se consideraba el canal H del espacio HSV y se aplicaba binarización (pasar a blanco y negro) haciendo uso de OTSU [1]. Esto se practicó con los canales L, A, B del espacio LAB [7], con los canales H, S y V, con los canales R, G B y con la imagen en escala de grises. En la figura 25 se muestra un ejemplo de esto. En la misma, la máscara la máscara señalada como kmeans es una máscara obtenida de aplicar el algoritmo kmeans con dos clusters y en el que los datos a segmentar son los vectores que tienen los valores HSV, LAB, RGB y de escala de grises y hay tantos datos como pixeles de la imagen. Esta máscara como se puede ver funciona bastante bien para la separación de la fruta aunque se puede comprobar que no funciona de forma perfecta en todos los casos. La máscara indicada como "color" es la máscara obtenida por la detección de colores antes mencionada.

Lo que se puede observar que sucede es que los fondos de las máscaras, en algunas de las ocasiones, son negros y en otros son blancos. En lugar de tomar las máscaras en sí, lo que se buscó es, con Sobel [2], los contornos y luego se volvió a binarizar con OTSU para

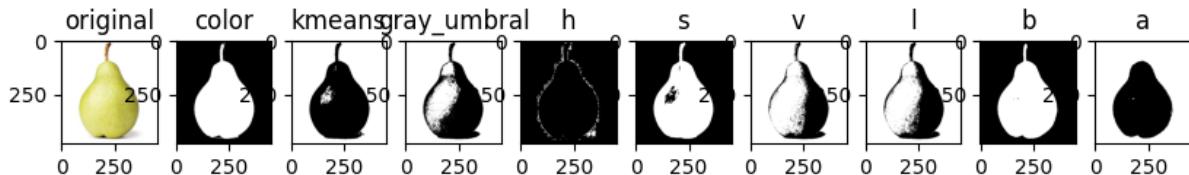


Figure 25: Máscaras probadas

resaltar solamente los contornos, como se muestra en la imagen 26 lo que permite pasar por alto la situación de qué color queda en el fondo. En esa imágen "Whole" representa la combinación de todos los contornos de las máscaras. Esta figura muestra que son varias las máscaras que pueden funcionar adecuadamente para separar a la fruta del fondo. Sin embargo esto es para frutas en las que el fondo de la imagen original es blanco o de otro color pero de tipo uniforme y sin texturas. En fondos mas complejos como por ejemplo, en una mesa con cierta textura, en cambio, en general no hay siquiera una máscara que permita separar correctamente a la fruta del fondo

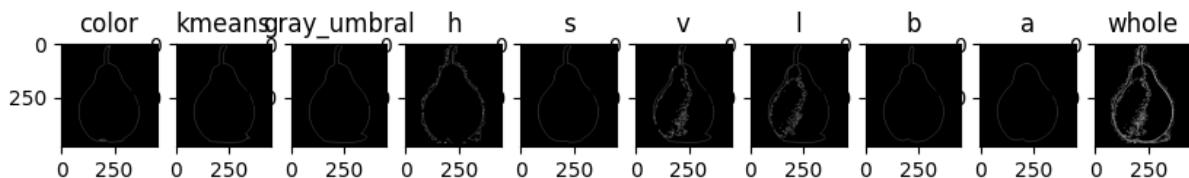


Figure 26: Contornos de las máscaras

Por esos motivos, finalmente se decidió utilizar solamente fondos blancos en las imágenes y se simplificó la separación de la fruta respecto del fondo.

El procesamiento final que se realiza en todas las imágenes para obtener las máscaras consiste en obtener los canales B del RGB, el canal S del espacio HSV y los canales A y B del espacio LAB, ya que se consideró que con estos se conseguían las máscaras más inertes ante sombras o variaciones de iluminación del entorno. Con esos datos por píxel, se procede a realizar la segmentación en dos clusters utilizando Kmeans de la biblioteca ‘sklearn.cluster’. Lo que sucede con Kmeans es que, en ocasiones, el fondo de la máscara obtenida por segmentación puede ser blanco o negro; sin embargo, lo que se pretende es un fondo negro en todos los casos, con la parte de la imagen donde se encuentra la fruta en blanco. Esto se resolvió evaluando los colores de los píxeles (blanco o negro) de la máscara en las esquinas y en las aristas, en matrices de píxeles con tamaños iguales a la 20-ava parte de la menor dimensión de la imagen. Luego, contando la cantidad de píxeles del total que se encuentran en blanco se puede determinar si el fondo de la máscara es blanco o negro. Si la cantidad es mayor al 75%, como quedó fijado, se considera que es fondo blanco y, por lo tanto, la máscara se invierte.

Con la máscara así definida, todavía se presentan algunos defectos en los bordes de la fruta como varias zonas pequeñas aisladas blancas, principalmente debido a los reflejos de la iluminación y la presencia de sombras. Para resolver esto, se aplican operaciones de erosión para limpiar la imagen y luego una operación de dilatación para expandir la máscara en la zona de la fruta y unir todos los bordes irregulares. De esta máscara

resultante, se obtiene el contorno. Este contorno luego se aproxima con líneas poligonales y se dibuja sobre una plantilla de fondo negro con relleno, de modo que todo en el interior del contorno es blanco y el exterior es negro. Esta es la máscara final que se aplicará posteriormente a la imagen para extraer las características y se guarda en un archivo.

En las figuras 27 y 28 se presentan como ejemplo una imagen y su máscara respectivamente.



Figure 27: Imagen Ejemplo



Figure 28: Máscara Ejemplo

#### 4.2.2 Extracción de Características

Las características utilizadas para la separación fueron el color de la fruta y los momentos de Hu 3 y 4.

- **Color:** Para obtener la característica de color, primero se definen rangos de color en el espacio HSV que correspondan a los colores verde, naranja, rojo y amarillo. Luego, se aplica la máscara obtenida durante el procesamiento a la imagen original, obteniendo la fruta en el fondo negro. Se determina en qué rango de color cae cada píxel y se cuenta la cantidad de píxeles que caen en cada rango. El color de la fruta en la imagen se determina como el rango de color más frecuente en la imagen.

Un problema que se tuvo en esta separación es que en ocasiones las manzanas eran clasificadas en el grupo de las naranjas por un límite no muy bien definido entre esos rangos de color. Para resolver ese problema, se aplicó una segunda condición aprovechando el hecho de que las otras frutas presentan en general muy poco nivel de rojo comparado con el color principal en sus cáscaras. De esta manera, para que una manzana no sea clasificada como naranja, además de determinar el primer color frecuente, se encuentra el segundo color frecuente. Si el primer color frecuente es naranja y el segundo color frecuente es el rojo, y además la cantidad de rojo es mayor a cierto porcentaje del color primero (se adoptó un 35%), entonces se acepta que en realidad el color de la fruta es rojo. De esta manera, solo las manzanas que erróneamente son clasificadas como de color naranja se clasifican de color rojo y ninguna naranja es considerada de color rojo.

- **Momentos de Hu:** Con los datos hasta ese momento, el color bastaba para la separación de los grupos. Sin embargo, se pretendió incorporar otras características que pudieran separar a los conjuntos cuando, por ejemplo, las frutas se presentaran descoloridas. Los momentos de Hu también se calculan aplicando primero la máscara a la imagen, se recorta la imagen al rectángulo que enmarca al contorno de la fruta, y luego se obtienen con la función `huMoments` de OpenCV.

#### 4.2.3 Clasificación con K-means

En cuanto al algoritmo K-means implementado, este utiliza la distancia euclídea como métrica. La inicialización de los centroides, aunque aleatoria, es "más inteligente", ya que hace uso de un algoritmo que aumenta las probabilidades de que los centroides estén suficientemente dispersos en el conjunto de datos, evitando grupos vacíos, por ejemplo. Cuando se utiliza esta forma de inicializar los centroides, el algoritmo se denomina K-means++ [8]. Básicamente, entre los datos, se eligen centroides de manera iterativa de modo tal que sea más probable elegir un dato como centroide cuanto más alejado esté de los centroides ya elegidos. Las imágenes 29, 30, 31 y 32 muestran la segmentación de las imágenes luego del entrenamiento.

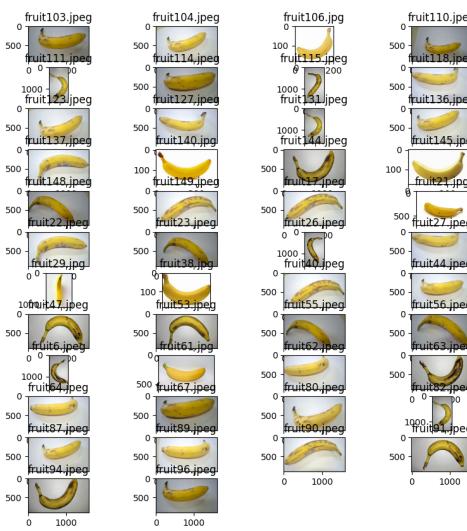


Figure 29: Cluster 0

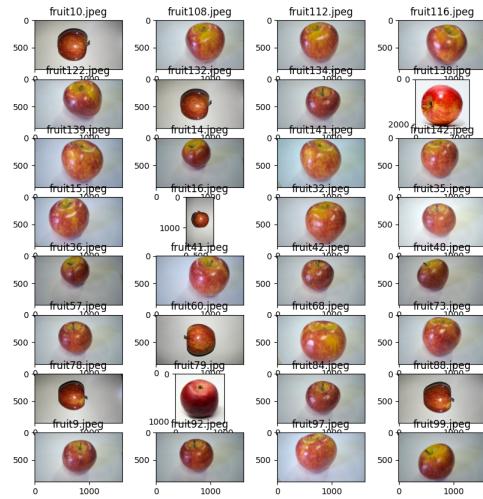


Figure 30: Cluster 1

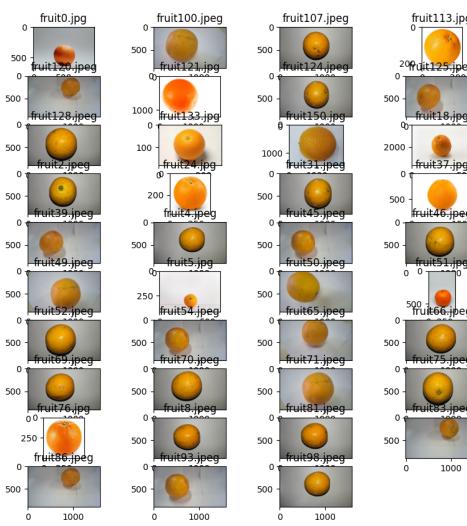


Figure 31: Cluster 2

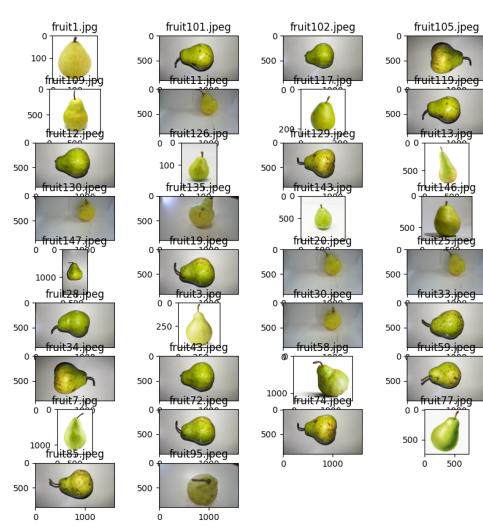


Figure 32: Cluster 3



#### 4.2.4 Clasificación

Luego del entrenamiento, como lo que se pretende hacer es identificar nuevas imágenes en base al modelo entrenado, es necesario poder determinar a cuál de los clústeres pertenece. Esto se logra con el algoritmo de clasificación Knn, donde se utiliza  $K = 1$ . Para ello, se toman las características o coordenadas de los centroides obtenidos en la base de datos del algoritmo durante el entrenamiento del segmentador Kmeans. Nuevamente, la métrica utilizada es la distancia euclíadiana. La etiquetación de los centroides se realiza de forma manual luego de evaluar la separación lograda con Kmeans. El algoritmo utilizado para Knn es el mismo utilizado para la clasificación de audios.

## 5 Ejemplo de Aplicación

### 5.1 Dataset

Por un lado, el dataset se encuentra en './dataset'. Dentro de la misma se encuentran las carpetas "audio" e "imagen" para los archivos de audio e imagen, respectivamente. En cada una de ellas se encuentran las carpetas "training", "validation" y "test", que contienen los datos de entrenamiento, los que se utilizan para la validación de los modelos y test que contiene los archivos de prueba del modelo. En el caso de las imágenes, esta última contiene las carpetas "shelf1", "shelf2", "shelf3" y "shelf4", para contener cada una, una foto de la fruta en el estante correspondiente. A su vez, dentro de las carpetas del último nivel se encuentran dos subcarpetas "original" y "processed", la primera contiene los archivos originales y la segunda alberga los archivos procesados (audios recortados o máscaras de imágenes).

### 5.2 Implementación

Las implementaciones de las distintas partes del agente se encuentran en la carpeta './implementation'. Dentro de la misma, la implementación del reconocimiento de voz se halla en la carpeta "audio/knn", y la de imagen en la carpeta "imagen/kmeans". En estas carpetas, el archivo "training.py" realiza el entrenamiento de los modelos correspondientes. En el caso del audio, extrae las características de los audios de entrenamiento, aplica la reducción de componentes y guarda los datos en el archivo "model.pkl", que además contendrá el objeto para aplicar las mismas operaciones de transformación de coordenadas sobre nuevos audios. Para la imagen, extrae las características de las imágenes de entrenamiento y entrena al clasificador. Los datos se guardan en el archivo "**training\_data.pkl**", que contendrá las características de las imágenes, los centroides obtenidos etiquetados y las imágenes de entrenamiento etiquetadas según el cluster de pertenencia.

Los archivos "validation.ipynb" prueban los modelos contra los datos contenidos en las correspondientes carpetas de validación. En el caso de imagen, un archivo adicional "processing.ipynb" permite obtener las máscaras de las imágenes del grupo de entrenamiento.

El archivo "testing.ipynb" para audio permite probar el clasificador contra audios grabados durante la ejecución del código y contra los audios cargados en la carpeta de testing.

### 5.2.1 Entrenamiento

**Audio:** Para entrenar el modelo de audio, se debe ejecutar el archivo de entrenamiento apretando "Run All" en el notebook (figura 33). El modelo se guardará y se mostrará en la última celda de código el agrupamiento obtenido (figura 34).

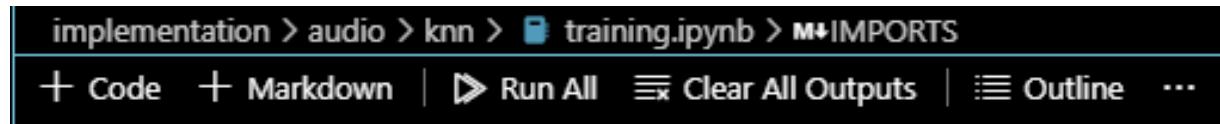


Figure 33: Ejecución

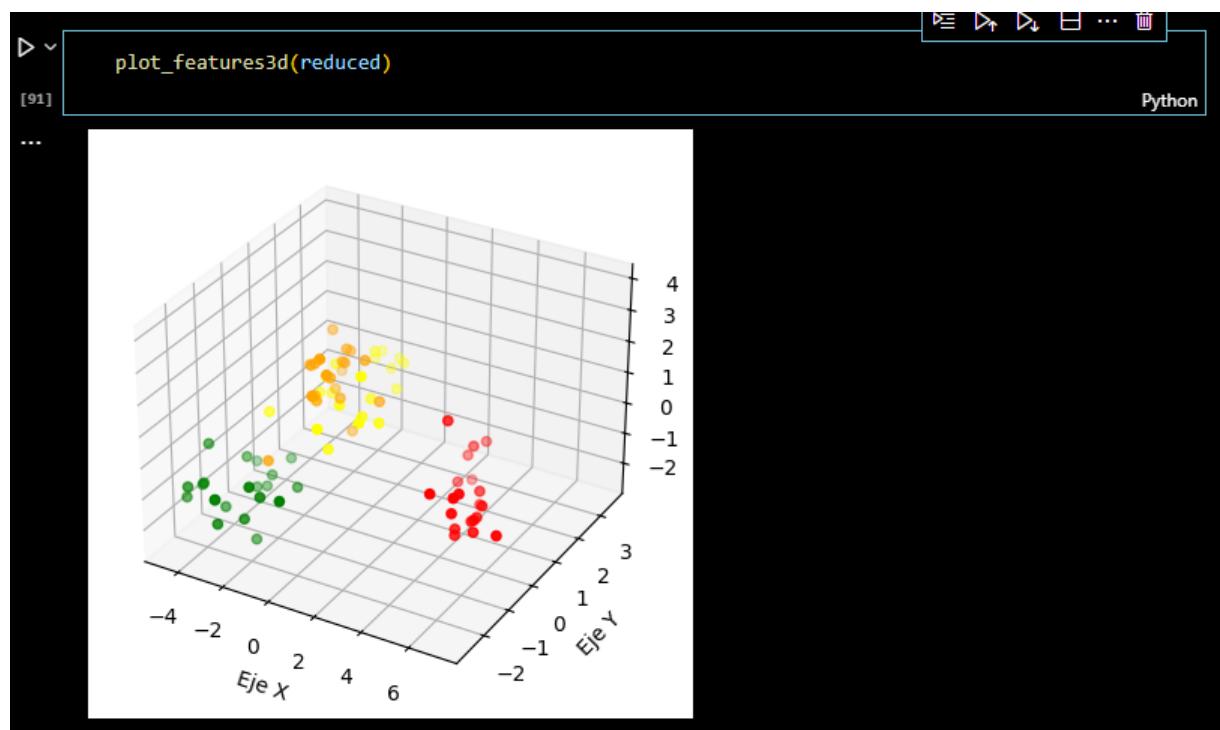


Figure 34: Agrupamiento obtenido

**Imagen:** Para el entrenamiento del modelo de imagen, se debe ejecutar primero el archivo de procesamiento "processing.ipynb" para obtener las máscaras de las imágenes. Luego, se debe ejecutar todo el código del archivo de entrenamiento. El modelo entrenado se guardará, se mostrarán los archivos en los clusters obtenidos y se mostrará la distribución de puntos en el espacio (figura 35).

### 5.2.2 Validación

Para la validación de cada modelo, hay que ejecutar el archivo de validación correspondiente. El mismo trabaja con los archivos en la carpeta de validación.

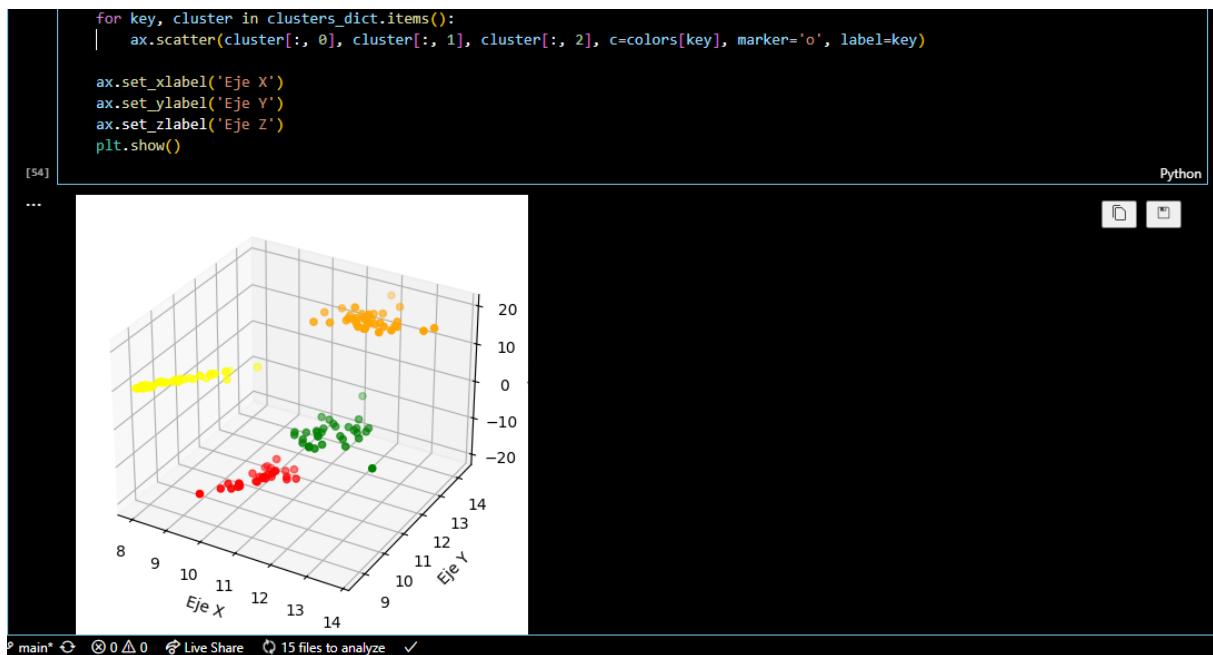


Figure 35: Entrenamiento reconocimiento de Imágen

**Reconocimiento de Audio:** Se muestra la ubicación de los puntos que representan los archivos de validación junto con los puntos del conjunto de entrenamiento en un gráfico (figura 36). Se indica para cada audio si se acertó la clasificación o no y se muestra el porcentaje de acierto (figura 37).

**Reconocimiento de Imagen:** Se clasifican las imágenes en la carpeta de validación con el modelo precalculado y se muestran las etiquetas predichas (figura 38).

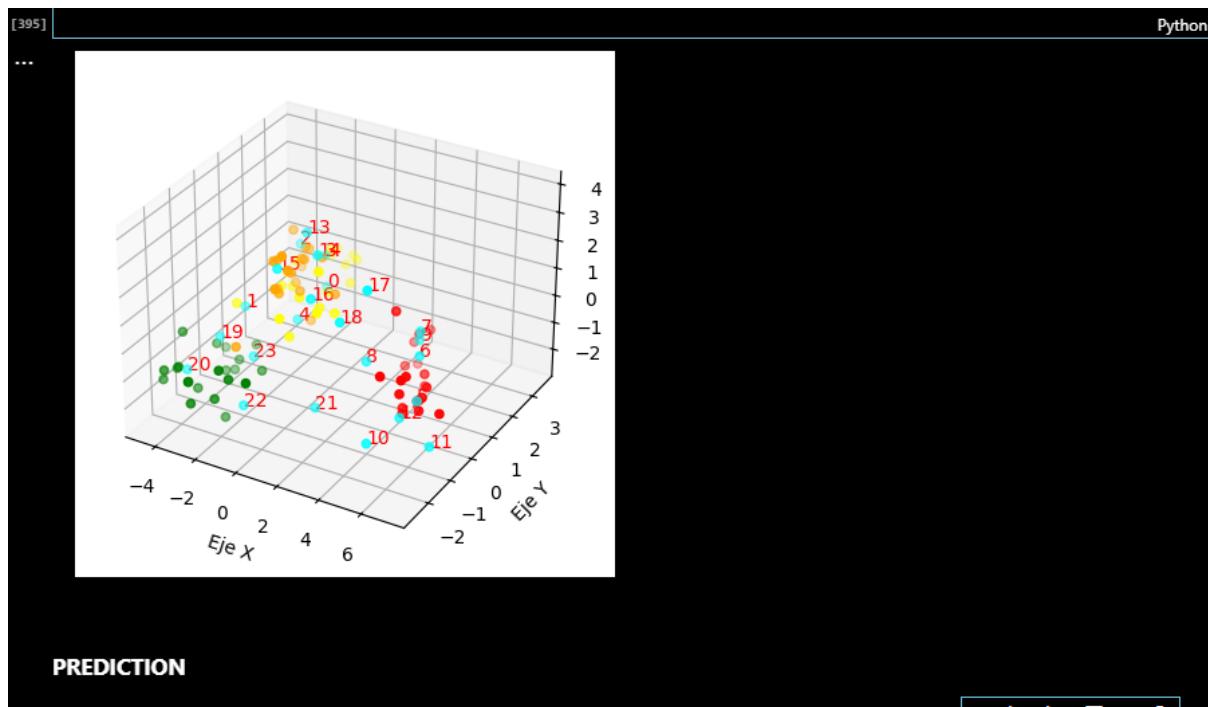
### 5.2.3 Solución

La solución final se encuentra en `'./implementation/solution/solution.py'`.

### 5.2.4 Ejemplo

Una vez entrenados los modelos según las indicaciones, sigue los siguientes pasos:

- Toma imágenes de las frutas en los estantes y colócalas en:  
`'./dataset/image/test/shelf/original'` según el estante correspondiente, colocando solo una en cada estante.
- Ejecuta la solución:
  - El programa indicará la etapa en que se encuentra (figure 39).
  - Luego, se solicitará al usuario una acción (figura 40).
  - Si el usuario presiona 'ENTER', comenzará la grabación y se indicará que se encuentra en proceso de grabación durante 2.5 segundos (figura 41). Al finalizar



```

+-----+
|   audios | banana1.wav | banana2.wav | banana3.wav | banana4.wav | banana5.wav | manzana1.wav | manzana2.wav |
+-----+
| prediction |   banana   |   banana   |   banana   |   banana   |   banana   |   manzana   |   manzana   |
| results    |   acierto  |
+-----+
Se acertaron: 24/24
El porcentaje de aciertos es: 100.0

```

Figure 36: Presentación gráfica en la validación

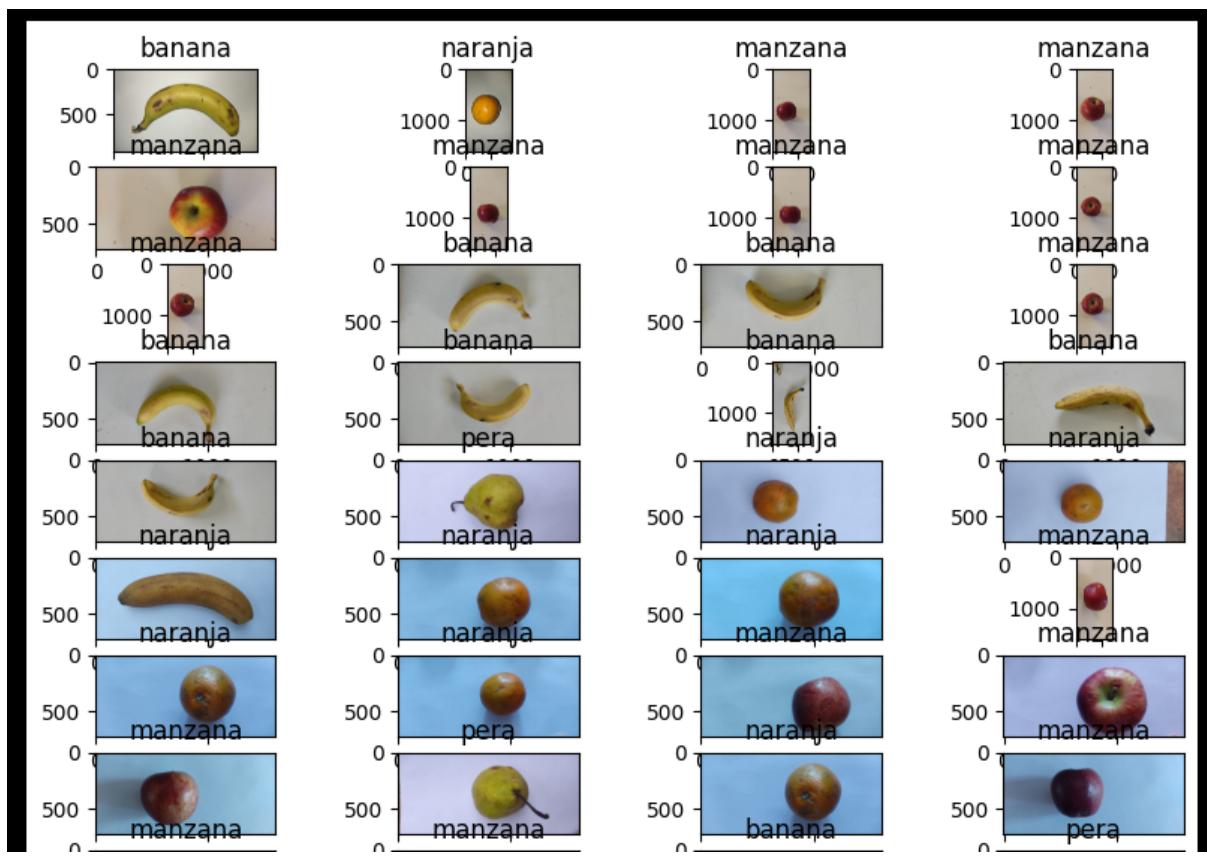


Figure 38: Máscaras probadas



Figure 39: Etapas del programa

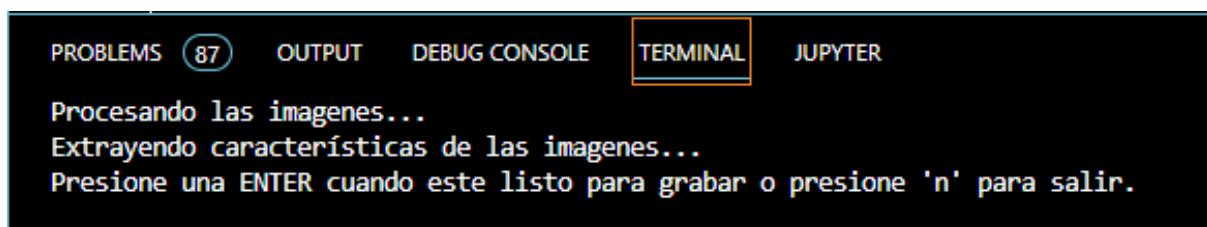


Figure 40: Prompt de usuario

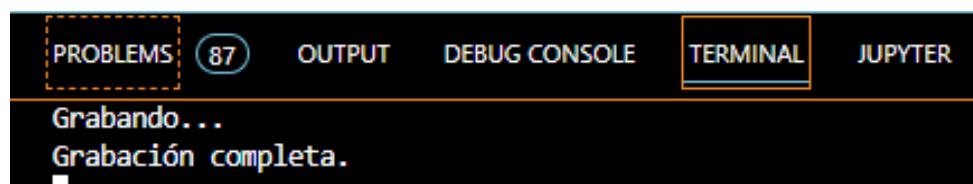
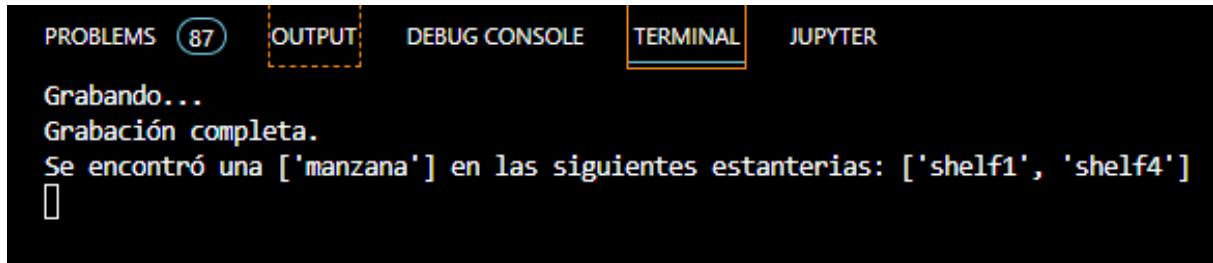


Figure 41: Grabación de audio.

la grabación, el archivo se guardará en la carpeta de archivos de prueba de audio y también el archivo procesado en las subcarpetas correspondientes.

- Al finalizar el procesamiento, se mostrará un mensaje indicando si se encontró la fruta y en qué estante o estantes (figura 42) y se desplegará una figura con la disposición de frutas en los estantes y un recuadro verde alrededor de la fruta identificada, si es que se encuentra (figura 43).



```

PROBLEMS 87 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
Grabando...
Grabación completa.
Se encontró una ['manzana'] en las siguientes estanterías: ['shelf1', 'shelf4']

```

Figure 42: Resultado ejecucion

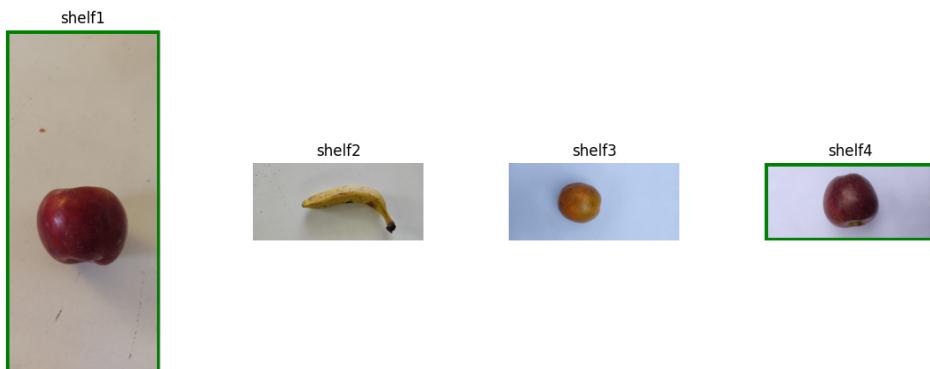


Figure 43: Presentación de los estantes

- Al cerrar la figura, se requerirá nuevamente la entrada del usuario hasta que decida terminar la ejecución.

## 6 Resultados

La evaluación de los resultados de la solución se hizo a través de la validación del modelo.

### 6.1 Reconocimiento de voz

Se incorporaron a la base de validación un total de 24 audios de todos los tipos de frutas etiquetados previamente y obtenidos de distintas personas. El modelo de reconocimiento de audio identificó correctamente el 100% de los audios, como queda puesto de manifiesto en el archivo de validación (figura 37). Otras pruebas se realizaron en diversas condiciones

y sin embargo se puede saber que el sistema no es perfecto, existen ciertas formas de pronunciar los nombres de las frutas que harán que no falle la predicción y tiene dificultades en la determinación de ciertas frutas como las naranjas a las que en general confunde con las bananas, cosa esperable por cuanto en el agrupamiento obtenido(figureas 22 Y 23) las bananas y las naranjas no se encuentran tan separadas.

## 6.2 Reconocimiento de Imagén

A la base de validación se incorporaron 47 imágenes de todos los tipos de frutas, todas en fondo blanco, en distintas posiciones, con iluminación natural en todos los casos. Se procuró utilizar frutas con el color mas vivo que haya sido posible. Los resultados se presentan en la imagen 44. Se puede obtener de esa imagen qué solamente identificó erróneamente a dos de las imágenes, ambas de una banana a las que identificó como naranjas. El porcentaje de acierto sería del 95%. De este experimento se puede determinar el peso que tiene la característica de color en la determinación del tipo de fruta y la poca contribución a la separación que enrealidad se logra con el uso de los momentos de Hu. Se puede notar que esas bananas mal identificadas tenían cierta cantidad de naranja en su cáscara y es muy probablemente esto lo que hizo que falle la identificación.

## 7 Conclusiones

En el desarrollo de este trabajo se exploraron diversas alternativas en la caracterización de datos de audio e imagen para tareas de clasificación y agrupamiento. Las principales dificultades surgieron al identificar las características más representativas del conjunto de datos, más que en la implementación de los algoritmos de clasificación (Knn) y segmentación (Kmeans). La investigación y las pruebas para encontrar un conjunto de características fueron tareas tediosas y arduas.

El conjunto de características extraídas, al menos para la caracterización del audio, terminó siendo en algunos casos demasiado específico, evaluando propiedades en ciertas partes a lo largo de la duración de un audio. En cuanto a las imágenes, la tarea que demandó más tiempo de prueba y trabajo fue la de separar la fruta del fondo para conservar solo sus características. Se encontró que era difícil lograr una buena separación cuando el fondo no era uniforme, y esta dificultad no se pudo resolver completamente, incluso probando diversas alternativas, incluida la posibilidad de utilizar algoritmos genéticos.

Sin embargo, bajo condiciones normalizadas tanto en el audio (pronunciación adecuada de los nombres y bajo nivel de ruido) como en la imagen (iluminación adecuada y frutas coloridas), se observó un reconocimiento generalmente satisfactorio. Se destaca la vulnerabilidad del sistema a variaciones en estas condiciones.

Por otro lado, la ejecución de los programas de entrenamiento requiere un tiempo considerable debido al procesamiento de datos, especialmente en el reconocimiento de imágenes. En cambio, la ejecución del programa principal es bastante rápida y sin mucha demora.

Tomando en consideración el tiempo invertido en el ajuste del sistema y la vulnerabilidad a variaciones en las entradas respecto del ideal, sería deseable contar con un sistema dotado de capacidades de aprendizaje automático. Este tipo de sistema tendría la capacidad de ajustarse de forma autónoma y asignar los pesos necesarios según convenga a las

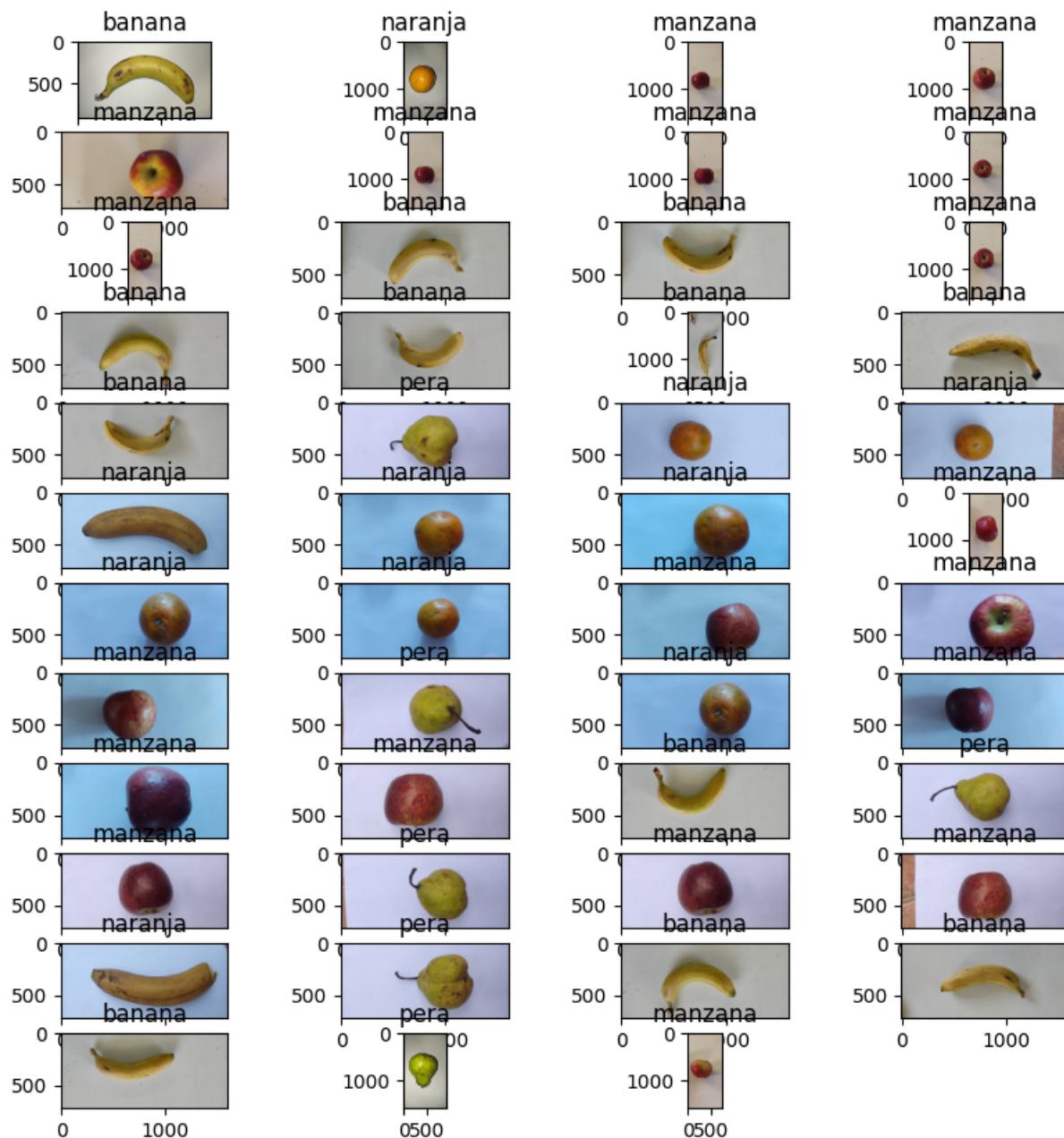


Figure 44: Resultados de la validación del Reconocedor de imágenes

distintas características extraídas de los datos.

Considero que explorar una solución de este tipo podría mejorar significativamente el rendimiento del agente en comparación con intentar mejorar el programa actual. Desarrollar una solución basada en aprendizaje automático podría proporcionar al sistema una mayor capacidad de generalización y adaptación a diversas condiciones, superando las limitaciones actuales, como la restricción a reconocer solo ciertas palabras y la falta de robustez frente a perturbaciones o variaciones, como el caso en el que una persona diga algo diferente a una de las frutas del problema actual y el sistema actual lo clasificaría de todas formas como una fruta.

## Referencias

- [1] Digital Sreeni. *Auto segmentation using multi-otsu*. 2020. URL: [https://www.youtube.com/watch?v=YdhhiXDQD14&list=PLZs0BAyNTZwZrTsptDOKRdSsV05mU5Qf0&index=20&ab\\_channel=DigitalSreeni](https://www.youtube.com/watch?v=YdhhiXDQD14&list=PLZs0BAyNTZwZrTsptDOKRdSsV05mU5Qf0&index=20&ab_channel=DigitalSreeni).
- [2] Digital Sreeni. *Edge Filters For Image Processing*. 2020. URL: [https://www.youtube.com/watch?v=Oy4duAOGdWQ&list=PLZs0BAyNTZwZrTsptDOKRdSsV05mU5Qf0&index=9&ab\\_channel=DigitalSreeni](https://www.youtube.com/watch?v=Oy4duAOGdWQ&list=PLZs0BAyNTZwZrTsptDOKRdSsV05mU5Qf0&index=9&ab_channel=DigitalSreeni).
- [3] Juan2000-coder. *Fruit Recognition GitHub Repository*. 2023. URL: [https://github.com/Juan2000-coder/fruit\\_recognition](https://github.com/Juan2000-coder/fruit_recognition).
- [4] musikalkemist. *AudioSignalProcessingForML*. 2020. URL: <https://github.com/musikalkemist/AudioSignalProcessingForML.git>.
- [5] Nombre del Canal o Usuario de YouTube. *Valerio Velardo - The Sound of AI*. 2020. URL: <https://youtube.com/playlist?list=PL-wATfeyAMNqIee7ch3q1bh4QJFAaeNv0&si=hv23f1Z0qVSXiwrh>.
- [6] Stuart Russell and Peter Norvig. “V:Learning”. In: *AI A Modern Approach*. Ed. by Michael Hirsch. Pearson, 2009, pp. 736–739.
- [7] Wikipedia. *Espacio de Color Lab*. URL: [https://es.wikipedia.org/wiki/Espacio\\_de\\_color\\_Lab](https://es.wikipedia.org/wiki/Espacio_de_color_Lab).
- [8] Wikipedia. *K-means++*. URL: <https://es.wikipedia.org/wiki/K-means%2B%2B>.

## PREPROCESAMIENTO DE LAS IMAGENES

- Obtención de máscaras.
- Obtención de contornos que separan fruta de fondo.
- Las máscaras se guardan en archivos de imagen en  
./dataset/images/training/processed
- Los contornos se guardan en ./implementation/images/kmeans/contornos.pkl

## LIBRERIAS

```
In [ ]: import os
import numpy as np
import cv2
from sklearn.cluster import KMeans
import joblib
```

## PATHS

```
In [ ]: image_path      = ' ../../dataset/images'
training_path    = os.path.join(image_path, 'training')
original_path    = os.path.join(training_path, 'original')
processed_path   = os.path.join(training_path, 'processed')
```

## LISTAS DE IMAGENES

```
In [ ]: original     = [os.path.join(original_path, image) for image in os.listdir(o
processed    = [os.path.join(processed_path, image) for image in os.listdir(p)]
```

## PROCESAMIENTO DE LAS IMAGENES

Al realizar la separación con el algoritmo kmeans en algunas ocasiones el fondo es blanco y en otras ocasiones el fondo es negro. Con la siguiente función se obtiene siempre una máscara con fondo negro y en blanco en la zona en donde se encuentra la fruta

```
In [ ]: def get_light_background(mask, f = 20, p = 0.75):
    height, width = mask.shape
    cluster_size  = min([height, width])//f
    cluster       = np.ones((cluster_size, cluster_size), np.uint8)

    # Corners
    corner1 = np.bitwise_and(cluster, mask[:cluster_size, :cluster_size])
    corner2 = np.bitwise_and(cluster, mask[:cluster_size:, -cluster_size:])
    corner3 = np.bitwise_and(cluster, mask[-cluster_size:, :cluster_size])
    corner4 = np.bitwise_and(cluster, mask[-cluster_size:, -cluster_size:])
    corners = [corner1, corner2, corner3, corner4]
```

```

# Sides
limitw1 = (width - cluster_size)//2
limitw2 = (width + cluster_size)//2
limith1 = (height - cluster_size)//2
limith2 = (height + cluster_size)//2

side1 = np.bitwise_and(cluster, mask[:cluster_size, limitw1:limitw2]
side2 = np.bitwise_and(cluster, mask[limith1:limith2, :cluster_size]
side3 = np.bitwise_and(cluster, mask[limith1:limith2, -cluster_size:]
side4 = np.bitwise_and(cluster, mask[-cluster_size:, limitw1:limitw2]
sides = [side1, side2, side3, side4]

# Determining the type of background
edges = corners + sides
light_background = sum(np.count_nonzero(edge) for edge in edges) > p*8

# Inverting if dark background
if light_background:
    return np.bitwise_not(mask)
return mask

```

*Obtencion de las máscaras y contornos para cada imagen*

```

In [ ]: for file in original:
    # BGR image
    image = cv2.imread(file)

    # Dimensions
    height, width, _ = image.shape

    # Pixel data vector
    data_vector = np.zeros((height * width, 4))

    # Obtener matrices del espacio de colores
    rgb_matrix = image.reshape((-1, 3))
    hsv_matrix = cv2.cvtColor(image, cv2.COLOR_BGR2HSV).reshape((-1, 3))
    lab_matrix = cv2.cvtColor(image, cv2.COLOR_BGR2LAB).reshape((-1, 3))

    # Asignar a la matriz de datos
    # Conservamos el canal G, S, A y B
    data_vector[:, 0] = rgb_matrix[:, 2]
    data_vector[:, 1] = hsv_matrix[:, 1]
    data_vector[:, 2:] = lab_matrix[:, 1:]

    # Segmentamos la imagen con los vectores obtenidos por cada pixel
    kmeans = KMeans(n_clusters = 2, n_init = 10) # 2 Clusters. Background
    kmeans.fit(data_vector)

    # Get clusters labels
    labels = kmeans.labels_

    # kmeans_mask
    kmeans_mask = labels.reshape(height, width)

```

```

kmeans_mask = kmeans_mask.astype(np.uint8) * 255

# Determinación del tipo de fondo de la máscara
kmeans_mask = get_light_background(kmeans_mask)

# Erosion y dilatación sobre la máscara
erosion_size      = min([height, width])//200
dilatacion_size   = min([height, width])//80
kernel_erosion    = np.ones((erosion_size,erosion_size), np.uint8)
eroded            = cv2.erode(kmeans_mask, kernel_erosion, iterations)
kernel_dilatacion = np.ones((dilatacion_size,dilatacion_size), np.uint8)
kmeans_mask       = cv2.dilate(eroded, kernel_dilatacion, iterations)

# Encontrar contornos
kmeans_cnt, _ = cv2.findContours(kmeans_mask, cv2.RETR_EXTERNAL, cv2.CC_COMPACT)
kmeans_cnt     = max(kmeans_cnt, key = cv2.contourArea)

# Contorno aproximado
epsilon         = 0.001 * cv2.arcLength(kmeans_cnt, True)
kmeans_cnt     = cv2.approxPolyDP(kmeans_cnt, epsilon, True)
kmeans_cnt     = (kmeans_cnt,)

# Template
tkmeans        = np.zeros((height, width), dtype=np.uint8)

# Dibujar
cv2.drawContours(tkmeans, kmeans_cnt, -1, 255, thickness = cv2.FILLED)

# Guardar mascara
cv2.imwrite(os.path.join(processed_path, os.path.basename(file)), tkmeans)

```

## ENTRENAMIENTO DEL SEGMENTADOR

- Se realiza extracción de características.
- Luego se aplica el algoritmo kmeans para agrupar las imágenes en clusters.
- Se guardan en un archivo:
  - Los centroides
  - Las características extraídas
  - La lista con la asignación de cada archivo a un cluster

## LIBRERIAS

```
In [ ]: import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import joblib
from scipy.spatial.distance import cdist
```

## PATHS

```
In [ ]: image_path      = ' ../../dataset/images'
training_path    = os.path.join(image_path, 'training')
original_path    = os.path.join(training_path, 'original')
processed_path   = os.path.join(training_path, 'processed')
training_data    = 'training_data.pkl'
```

## LISTAS DE IMAGENES

```
In [ ]: original    = [os.path.join(original_path, image) for image in os.listdir(original_path)]
processed   = [os.path.join(processed_path, image) for image in os.listdir(processed_path)]
```

## KMEANS

```
In [ ]: def kmeans(n_clusters, features, tol = 1e-4, max_iter = 300):

    # kmeans++.
    # Selección de los centroides iniciales
    centroids = [features[np.random.choice(features.shape[0])]]

    for _ in range(1, n_clusters):
        # Calcular las distancias cuadradas desde los centroides actuales
        dist_sq = np.array([min([np.linalg.norm(c - x)**2 for c in centroids])
                           for x in features])

        # Calcular las probabilidades de elegir cada punto como próximo centroide
        probabilities = dist_sq / np.sum(dist_sq)

        # Elegir el próximo centroide usando las probabilidades
        next_centroid = features[np.random.choice(features.shape[0], p = probabilities)]
```

```

        centroids.append(next_centroid)

centroids = np.vstack(centroids)

# Iteración
for _ in range(max_iter):
    # Clusters
    clusters      = []
    labels        = np.empty(features.shape[0])
    new_centroids = []

    # Distancias de cada punto a cada centroide
    dist         = cdist(centroids, features)
    sorted_ind   = np.argsort(dist, axis = 0)

    # Construcción de los clusters
    # Y recálculo de los centroides

    for j in range(n_clusters):
        index      = sorted_ind[0, :] == j
        cluster    = features[index, :]
        labels[index] = j
        clusters.append(cluster)
        centroid   = np.mean(cluster, axis = 0)
        new_centroids.append(centroid)
    new_centroids = np.vstack(new_centroids)

    # Verificamos condición de parada por tolerancia
    dist = np.linalg.norm(centroids - new_centroids, axis = 1)
    if np.max(dist) < tol:
        break

    centroids = new_centroids
# Devolvemos la matriz que tiene por filas los centroides.

# Devolvemos la lista de labels que indican la pertenencia a un clúster
return list(labels), clusters, centroids

```

## RANGOS DE COLOR

```

In [ ]: lower_red_2 = np.array([170, 60, 60])
upper_red_2 = np.array([179, 255, 255])

lower_red_1 = np.array([0, 60, 60])
upper_red_1 = np.array([8, 255, 255])

lower_orange = np.array([8, 120, 80])
upper_orange = np.array([21, 255, 255])

lower_yellow = np.array([21, 50, 80])
upper_yellow = np.array([25, 255, 255])

```

```
lower_green = np.array([25, 40, 40])
upper_green = np.array([100, 255, 255])
```

## EXTRACCIÓN DE CARACTERÍSTICAS

```
In [ ]: conversion_color = {'V' :-20, 'R' : -10, 'A' : 10, 'N' : 20}
names          = [os.path.basename(file) for file in original]
image_features = dict.fromkeys(names)

for image_file, mask_file in zip(original, processed):
    # Leer la imagen y la máscara
    image = cv2.imread(image_file)
    mask  = cv2.imread(mask_file, cv2.IMREAD_GRAYSCALE)

    # Convertir la imagen de BGR a HSV
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Aplicar la máscara
    fruit = cv2.bitwise_and(hsv_image, hsv_image, mask=mask)

    #-----Extracción de los momentos de Hu-----

    # Encontrar el rectángulo delimitador de la fruta
    (x, y, w, h) = cv2.boundingRect(mask)

    # Recortar la imagen original para obtener solo la región de la fruta
    trimed = fruit[y:y + h, x:x + w]

    # Convertir la imagen a escala de grises si es necesario
    trimed_gray = cv2.cvtColor(trimed, cv2.COLOR_BGR2GRAY)

    # Calcular los momentos de la imagen
    momentos = cv2.moments(trimed_gray)

    # Calcular los momentos de Hu
    momentos_hu = cv2.HuMoments(momentos)

    # Aplicar Logaritmo a los momentos de Hu para mejorar la escala
    log_moments_hu = -np.sign(momentos_hu) * np.log10(np.abs(momentos_hu))
    moments = log_moments_hu.reshape(-1)

    #-----Extracción de color-----
    conteo = {
        'V' : np.sum(np.all(np.logical_and(lower_green <= fruit, fruit <=
        'R1': np.sum(np.all(np.logical_and(lower_red_1 <= fruit, fruit <=
        'R2': np.sum(np.all(np.logical_and(lower_red_2 <= fruit, fruit <=
        'A' : np.sum(np.all(np.logical_and(lower_yellow <= fruit, fruit <=
        'N' : np.sum(np.all(np.logical_and(lower_orange <= fruit, fruit <=
    }
    conteo_por_rango = {
        'V': conteo['V'],
        'R': conteo['R1'] + conteo['R2'],
        'A': conteo['A'],
```

```

'N': conteo['N']
}

sorted_conteo = sorted(conteo_por_rango.items(), key=lambda x: x[1], reverse=True)

# Obtener el segundo elemento más grande
segundo_mas_grande = sorted_conteo[1]

# Obtener la etiqueta y el valor del segundo elemento más grande
etiqueta_segundo_mas_grande = segundo_mas_grande[0]
valor_segundo_mas_grande = segundo_mas_grande[1]

# Obtener la etiqueta basándose en el rango con el mayor conteo
etiqueta = max(conteo_por_rango, key = conteo_por_rango.get)

# Se usa el hecho de que a excepción de las manzanas, el resto de las
if (etiqueta_segundo_mas_grande == 'R') and (valor_segundo_mas_grande > 1):
    etiqueta = 'R'

color = conversion_color[etiqueta]

-----Vector de características-----
image_features[os.path.basename(image_file)] = np.append(moments[2:4],

```

## APLICACION DE KMEANS

```
In [ ]: # Obtener los valores de circularidad como un array de NumPy
features = np.vstack(list(image_features.values()))

# Especificar el número de clusters (k)
num_clusters = 4

# Aplicamos kmeans
labels, clusters, centroids = kmeans(num_clusters, features)
clusters_dict = dict(zip([0, 1, 2, 3], clusters)) # Para representarlos
```

## REPRESENTACIÓN DE LOS CLUSTERS

Agrupamos las imágenes en los clusters

```
In [ ]: labels_dict = dict(zip(names, [int(label) for label in labels]))
image_clusters = dict.fromkeys(set(labels))
for file, label in labels_dict.items():
    if image_clusters[label] is None:
        image_clusters[label] = []
    image_clusters[label].append(file)
```

Representación de las imágenes

```
In [ ]: for key, cluster in image_clusters.items():
    print(key)
```

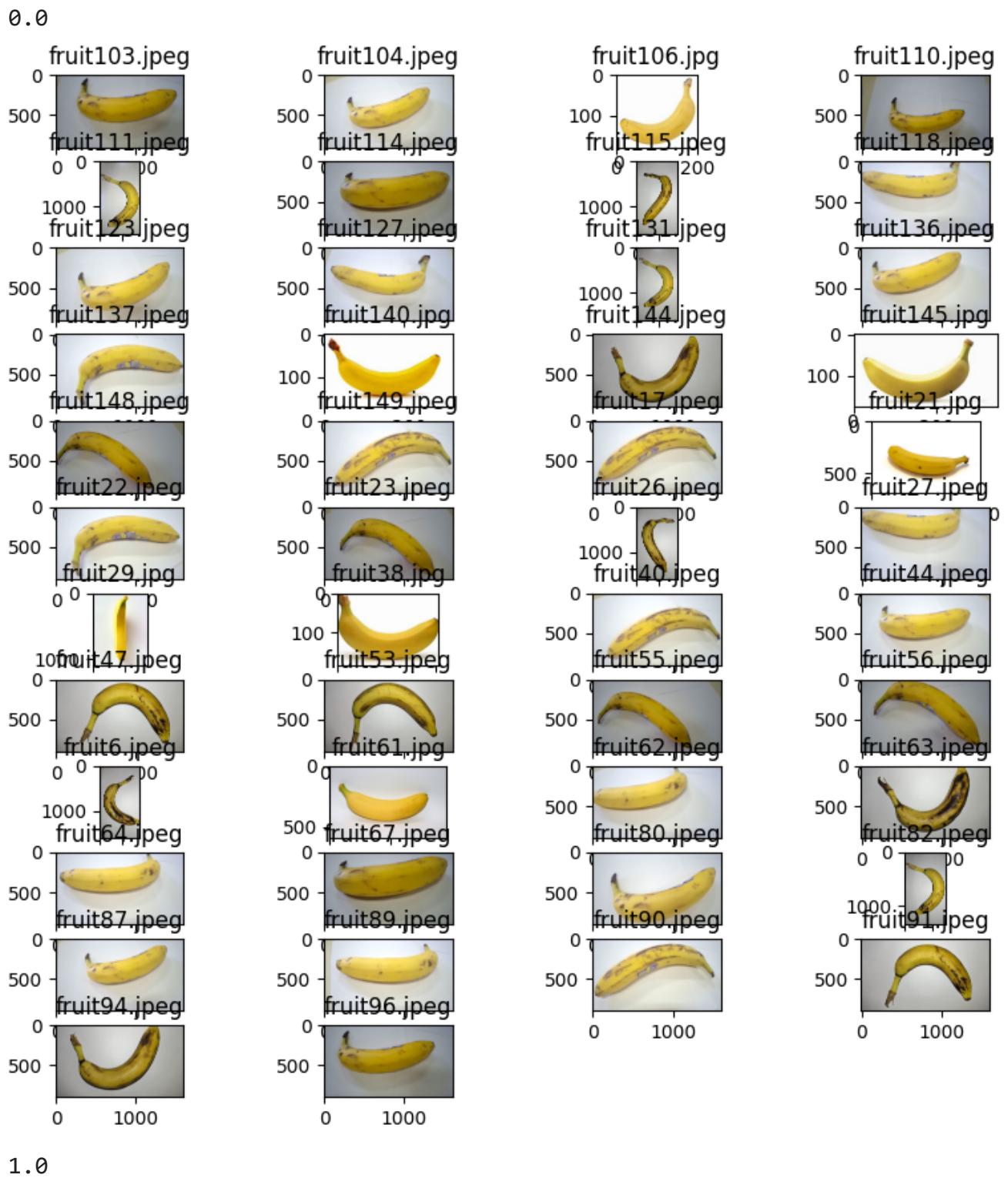
```

cols = 4
rows = len(cluster)//cols

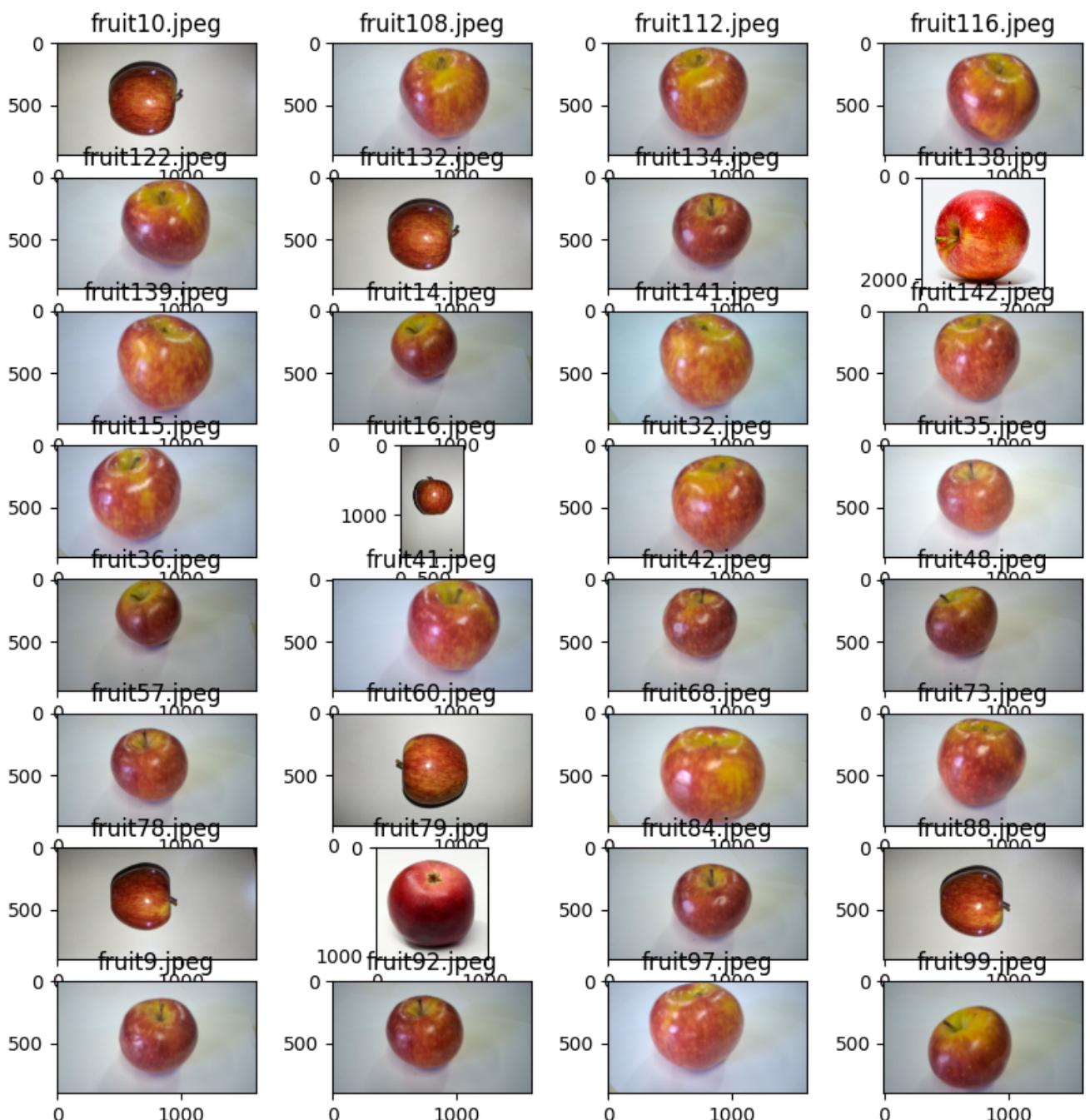
if len(cluster)%cols != 0:
    rows += 1

plt.figure(figsize = (10, 10))
for i, element in enumerate(cluster):
    file = os.path.join(original_path, element)
    plt.subplot(rows, cols, i + 1)
    plt.imshow(cv2.cvtColor(cv2.imread(file), cv2.COLOR_BGR2RGB))
    plt.title(os.path.basename(file))
plt.show()

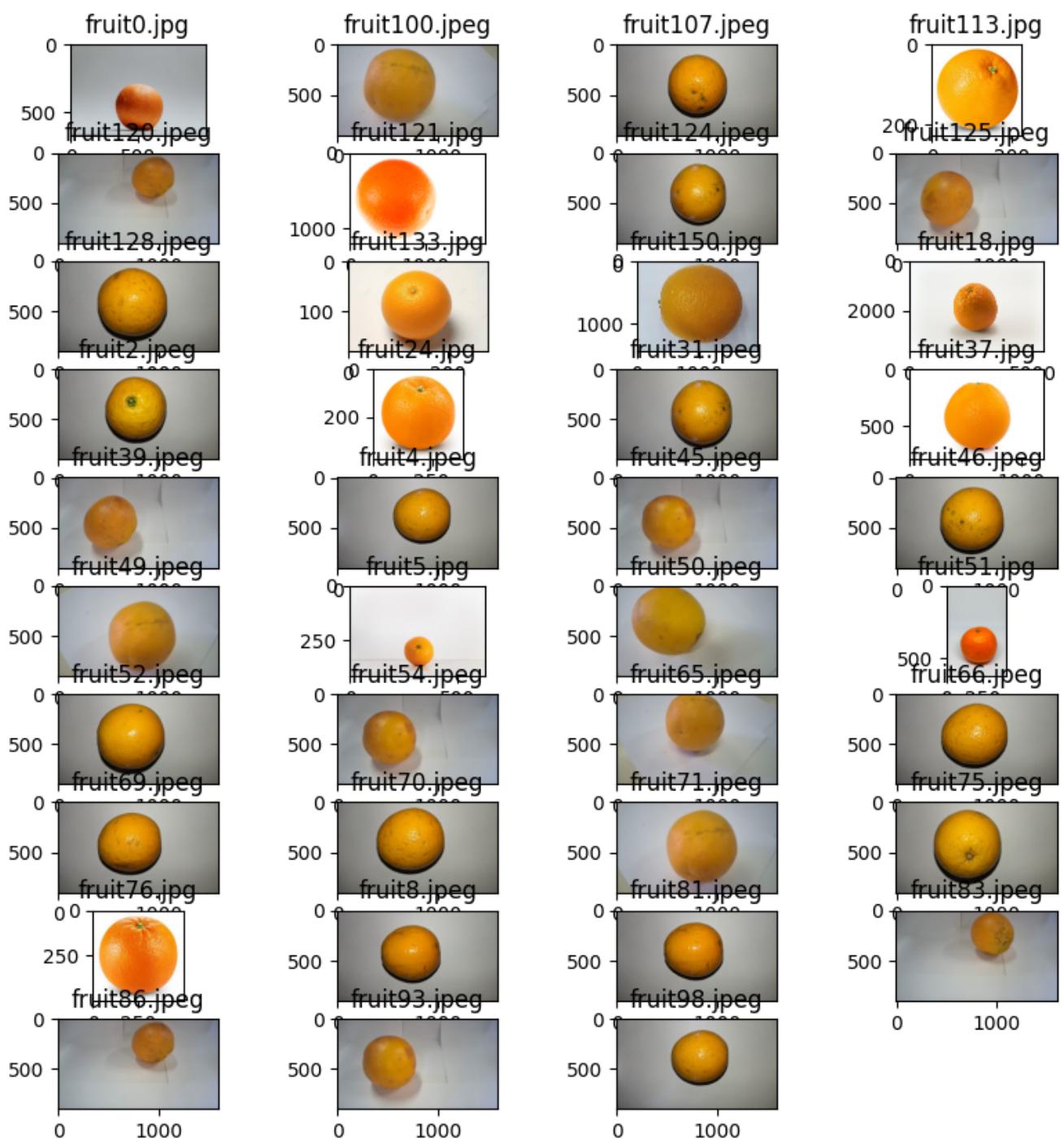
```



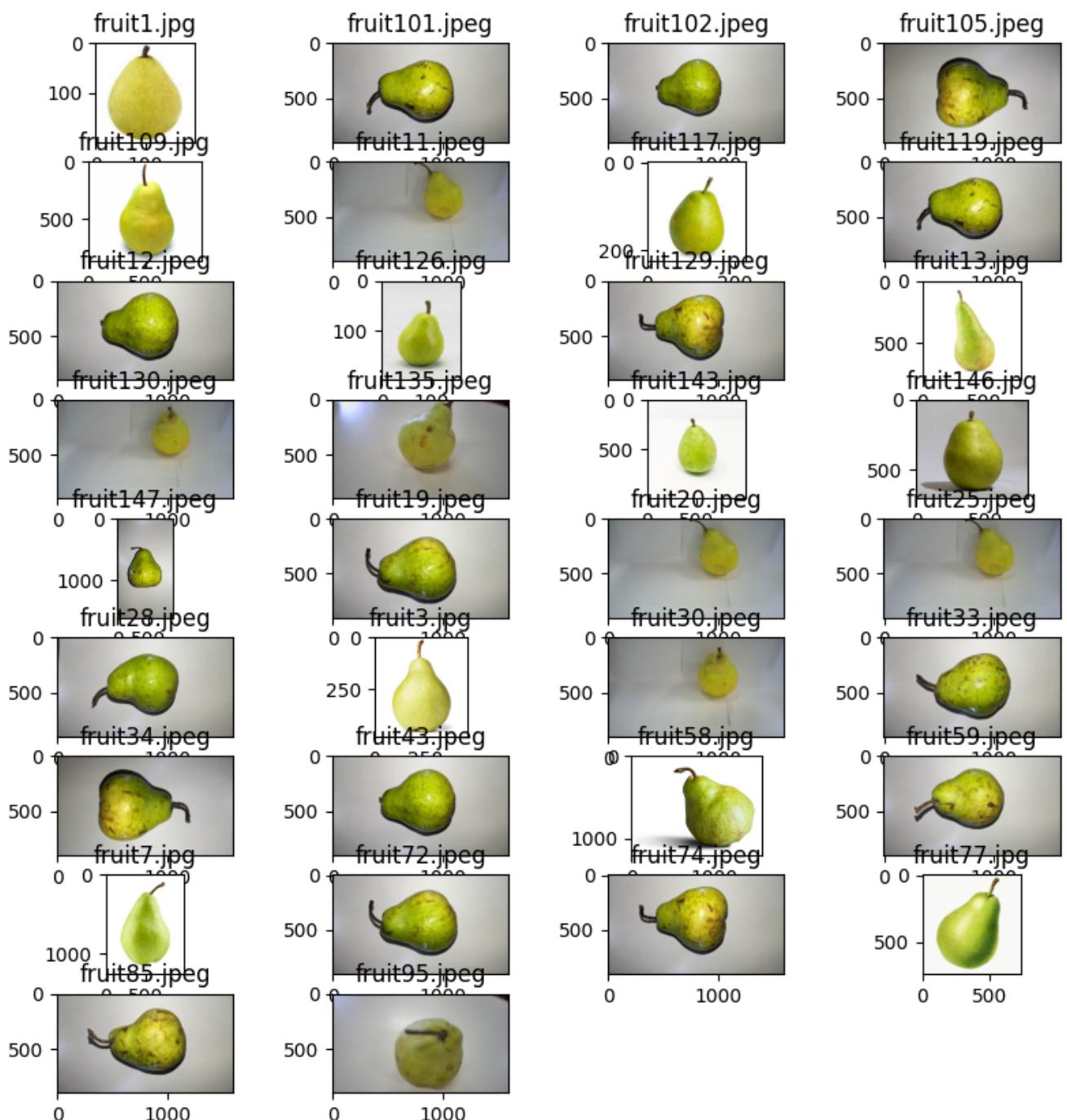
1.0



2.0



3 .0

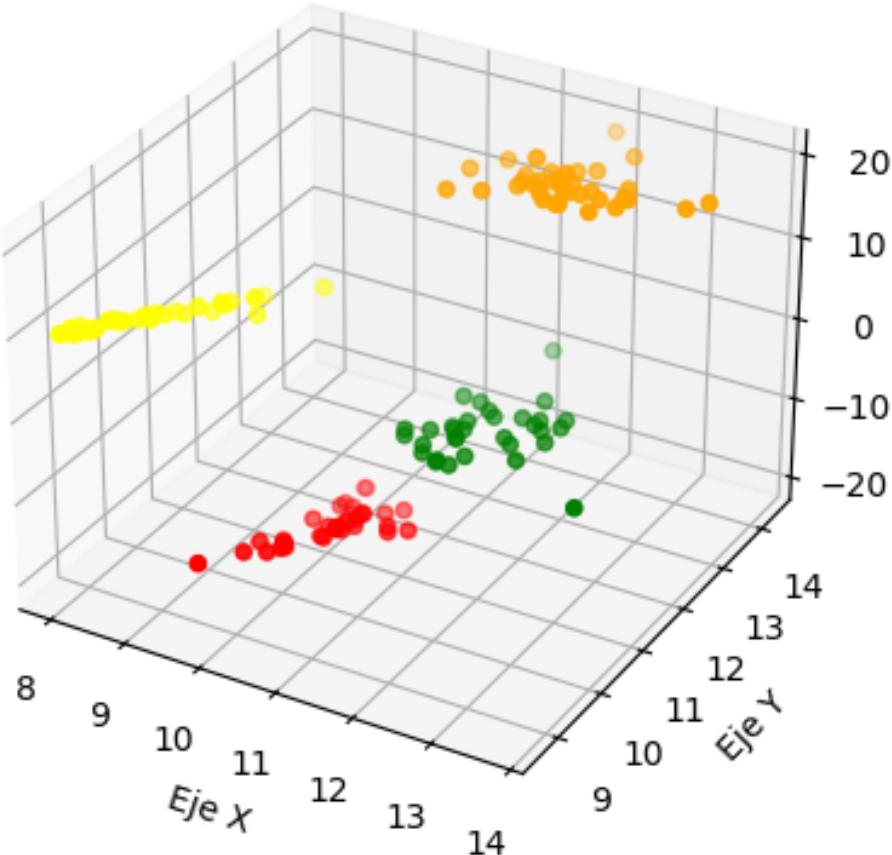


## REPRESENTACIÓN DE LOS PUNTOS EN EL ESPACIO

```
In [ ]: #3d
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
colors = dict(zip(clusters_dict.keys(), ['yellow','green','orange','red']))

for key, cluster in clusters_dict.items():
    ax.scatter(cluster[:, 0], cluster[:, 1], cluster[:, 2], c=colors[key])

ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')
plt.show()
```



## OBTENEMOS POR OBSERVACIÓN LAS ETIQUETAS DE CADA CLUSTER Y LAS DE LOS CENTROIDES

```
In [ ]: numeric2fruit = {0: 'banana', 1:'manzana', 2:'naranja', 3:'pera'}
```

*Pasamos de labels numéricas a nombres de frutas*

```
In [ ]: numeric_labels = [int(label) for label in labels]
fruit_labels = []
for i, label in enumerate(numeric_labels):
    fruit_labels.append(numeric2fruit[label])
```

*Etiquetamos los centroides*

```
In [ ]: labeled_centroids = dict()
for i, centroid in enumerate(centroids):
    labeled_centroids[numeric2fruit[i]] = centroid.reshape(1,-1)
```

## GUARDAMOS LOS DATOS EN EL ARCHIVO DE DATA

```
In [ ]: # Guardamos los labels también como diccionario
labels_dict = dict(zip(names, fruit_labels))
data = {'features': image_features, 'labels': labels_dict, 'centroids': centroids}
joblib.dump(data, training_data)
```

```
Out[ ]: ['training_data.pkl']
```

## VALIDACION DEL MODELO

- Basicamente utilizando el algoritmo de nearest neighbors (k nearest neighbors con k = 1), en donde los puntos dados son los centroides calculados durante el entrenamiento, se busca clasificar una nueva fruta en el cluster de uno de los centroides.

## LIBRERIAS

```
In [ ]: import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import joblib
from scipy.spatial.distance import cdist
```

## PATHS

```
In [ ]: image_path      = '../.../dataset/images'
validation_path   = os.path.join(image_path, 'validation')
original_path     = os.path.join(validation_path, 'original')
processed_path    = os.path.join(validation_path, 'processed')
training_data     = 'training_data.pkl'
```

## LISTAS DE IMAGENES

```
In [ ]: original    = [os.path.join(original_path, image) for image in os.listdir(o
```

## PROCESAMIENTO DE LAS IMÁGENES

```
In [ ]: def get_light_background(mask, f = 20, p = 0.75):
    height, width = mask.shape
    cluster_size  = min([height, width])//f
    cluster       = np.ones((cluster_size, cluster_size), np.uint8)

    # Corners
    corner1 = np.bitwise_and(cluster, mask[:cluster_size, :cluster_size])
    corner2 = np.bitwise_and(cluster, mask[:cluster_size:, -cluster_size:])
    corner3 = np.bitwise_and(cluster, mask[-cluster_size:, :cluster_size])
    corner4 = np.bitwise_and(cluster, mask[-cluster_size:, -cluster_size:])
    corners = [corner1, corner2, corner3, corner4]

    # Sides
    limitw1 = (width - cluster_size)//2
    limitw2 = (width + cluster_size)//2
    limith1 = (height - cluster_size)//2
    limith2 = (height + cluster_size)//2
```

```

side1 = np.bitwise_and(cluster, mask[:cluster_size, limitw1:limitw2]
side2 = np.bitwise_and(cluster, mask[limith1:limith2, :cluster_size]
side3 = np.bitwise_and(cluster, mask[limith1:limith2, -cluster_size:]
side4 = np.bitwise_and(cluster, mask[-cluster_size:, limitw1:limitw2]
sides = [side1, side2, side3, side4]

# Determining the type of background
edges = corners + sides
light_background = sum(np.count_nonzero(edge) for edge in edges) > p*8

# Inverting if dark background
if light_background:
    return np.bitwise_not(mask)
return mask

```

```

In [ ]: for file in original:
# BGR image
image = cv2.imread(file)

# Dimensions
height, width, _ = image.shape

# Pixel data vector
data_vector = np.zeros((height * width, 4))

# Obtener matrices del espacio de colores
rgb_matrix = image.reshape((-1, 3))
hsv_matrix = cv2.cvtColor(image, cv2.COLOR_BGR2HSV).reshape((-1, 3))
lab_matrix = cv2.cvtColor(image, cv2.COLOR_BGR2LAB).reshape((-1, 3))

# Asignar a la matriz de datos
# Conservamos el canal G, S, A y B
data_vector[:, 0] = rgb_matrix[:, 2]
data_vector[:, 1] = hsv_matrix[:, 1]
data_vector[:, 2:] = lab_matrix[:, 1:]

# Segmentamos la imagen con los vectores obtenidos por cada pixel
kmeans = KMeans(n_clusters = 2, n_init = 10, random_state=42) # 2 Clusters
kmeans.fit(data_vector)

# Get clusters labels
labels = kmeans.labels_

# kmeans_mask
kmeans_mask = labels.reshape(height, width)
kmeans_mask = kmeans_mask.astype(np.uint8) * 255

# Determinación del tipo de fondo de la máscara
kmeans_mask = get_light_background(kmeans_mask)

# Erosion y dilatación sobre la máscara
erosion_size = min([height, width])//200
dilatacion_size = min([height, width])//80
kernel_erosion = np.ones((erosion_size, erosion_size), np.uint8)

```

```

eroded          = cv2.erode(kmeans_mask, kernel_erosion, iterations
kernel_dilatacion = np.ones((dilatacion_size,dilatacion_size), np.uint8)
kmeans_mask      = cv2.dilate(eroded, kernel_dilatacion, iterations

# Encontrar contornos
kmeans_cnt, _ = cv2.findContours(kmeans_mask, cv2.RETR_EXTERNAL, cv2.CC_COMPACT)
kmeans_cnt     = max(kmeans_cnt, key = cv2.contourArea)

# Contorno aproximado
epsilon        = 0.001 * cv2.arcLength(kmeans_cnt, True)
kmeans_cnt     = cv2.approxPolyDP(kmeans_cnt, epsilon, True)
kmeans_cnt     = (kmeans_cnt,)

# Template
tkmeans        = np.zeros((height, width), dtype=np.uint8)

# Dibujar
cv2.drawContours(tkmeans, kmeans_cnt, -1, 255, thickness = cv2.FILLED)

# Guardar mascara
cv2.imwrite(os.path.join(processed_path, os.path.basename(file)), tkmeans)

```

In [ ]: processed = [os.path.join(processed\_path, image) for image in os.listdir(processed\_path)]

## RANGOS DE COLOR

```

In [ ]: lower_red_2 = np.array([170, 60, 60])
upper_red_2 = np.array([179, 255, 255])

lower_red_1 = np.array([0, 60, 60])
upper_red_1 = np.array([8, 255, 255])

lower_orange = np.array([8, 120, 80])
upper_orange = np.array([21, 255, 255])

lower_yellow = np.array([21, 50, 80])
upper_yellow = np.array([25, 255, 255])

lower_green = np.array([25, 40, 40])
upper_green = np.array([100, 255, 255])

```

## EXTRACCIÓN DE CARACTERÍSTICAS

```

In [ ]: conversion_color = {'V' :-20, 'R' : -10, 'A' : 10, 'N' : 20}
names           = [os.path.basename(file) for file in original]
image_features = dict.fromkeys(names)

for image_file, mask_file in zip(original, processed):
    # Leer la imagen y la máscara
    image = cv2.imread(image_file)
    mask = cv2.imread(mask_file, cv2.IMREAD_GRAYSCALE)

    # Convertir la imagen de BGR a HSV

```

```

hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Aplicar la máscara
fruit = cv2.bitwise_and(hsv_image, hsv_image, mask=mask)

-----Extracción de los momentos de Hu-----

# Encontrar el rectángulo delimitador de la fruta
(x, y, w, h) = cv2.boundingRect(mask)

# Recortar la imagen original para obtener solo la región de la fruta
trimed = fruit[y:y + h, x:x + w]

# Convertir la imagen a escala de grises si es necesario
trimed_gray = cv2.cvtColor(trimed, cv2.COLOR_BGR2GRAY)

# Calcular los momentos de la imagen
momentos = cv2.moments(trimed_gray)

# Calcular los momentos de Hu
momentos_hu = cv2.HuMoments(momentos)

# Aplicar Logaritmo a los momentos de Hu para mejorar la escala
log_moments_hu = -np.sign(momentos_hu) * np.log10(np.abs(momentos_hu))
moments = log_moments_hu.reshape(-1)

-----Extracción de color-----
conteo = {
    'V' : np.sum(np.logical_and(lower_green <= fruit, fruit <=
        'R1') : np.sum(np.logical_and(lower_red_1 <= fruit, fruit <=
        'R2') : np.sum(np.logical_and(lower_red_2 <= fruit, fruit <=
        'A' : np.sum(np.logical_and(lower_yellow <= fruit, fruit <=
        'N' : np.sum(np.logical_and(lower_orange <= fruit, fruit <=
    }
conteo_por_rango = {
    'V': conteo['V'],
    'R': conteo['R1'] + conteo['R2'],
    'A': conteo['A'],
    'N': conteo['N']
}

sorted_conteo = sorted(conteo_por_rango.items(), key=lambda x: x[1], reverse=True)

# Obtener el segundo elemento más grande
segundo_mas_grande = sorted_conteo[1]

# Obtener la etiqueta y el valor del segundo elemento más grande
etiqueta_segundo_mas_grande = segundo_mas_grande[0]
valor_segundo_mas_grande = segundo_mas_grande[1]

# Obtener la etiqueta basándose en el rango con el mayor conteo
etiqueta = max(conteo_por_rango, key = conteo_por_rango.get)

# Se usa el hecho de que a excepción de las manzanas, el resto de las

```

```

if (etiqueta_segundo_mas_grande == 'R')and(valor_segundo_mas_grande >
    etiqueta = 'R'

color = conversion_color[etiqueta]

#-----Vector de características-----
image_features[os.path.basename(image_file)] = np.append(moments[2:4],

```

## RECUPERAMOS LOS CENTROIDES Y APLICACIÓN DE KNN

```
In [ ]: def knn(training, test, k_n):
    X           = np.concatenate([v for v in training.values()], axis = 0)
    y           = np.concatenate([[k] * v.shape[0] for k, v in training.it
    dist        = cdist(test, X)
    sorted_ind  = np.argsort(dist, axis = 1)
    sorted_k    = sorted_ind[:, 0:k_n]
    predicted   = []

    for row in sorted_k:
        labels     = list(y[row])
        prediction = max(set(labels), key = labels.count)
        predicted.append(prediction)
    return predicted
```

```
In [ ]: data           = joblib.load(training_data)
centroids      = data['centroids']
training_features = data['features']
training_labels = data['labels']
prediction     = knn(centroids, np.vstack(list(image_features.values())))
```

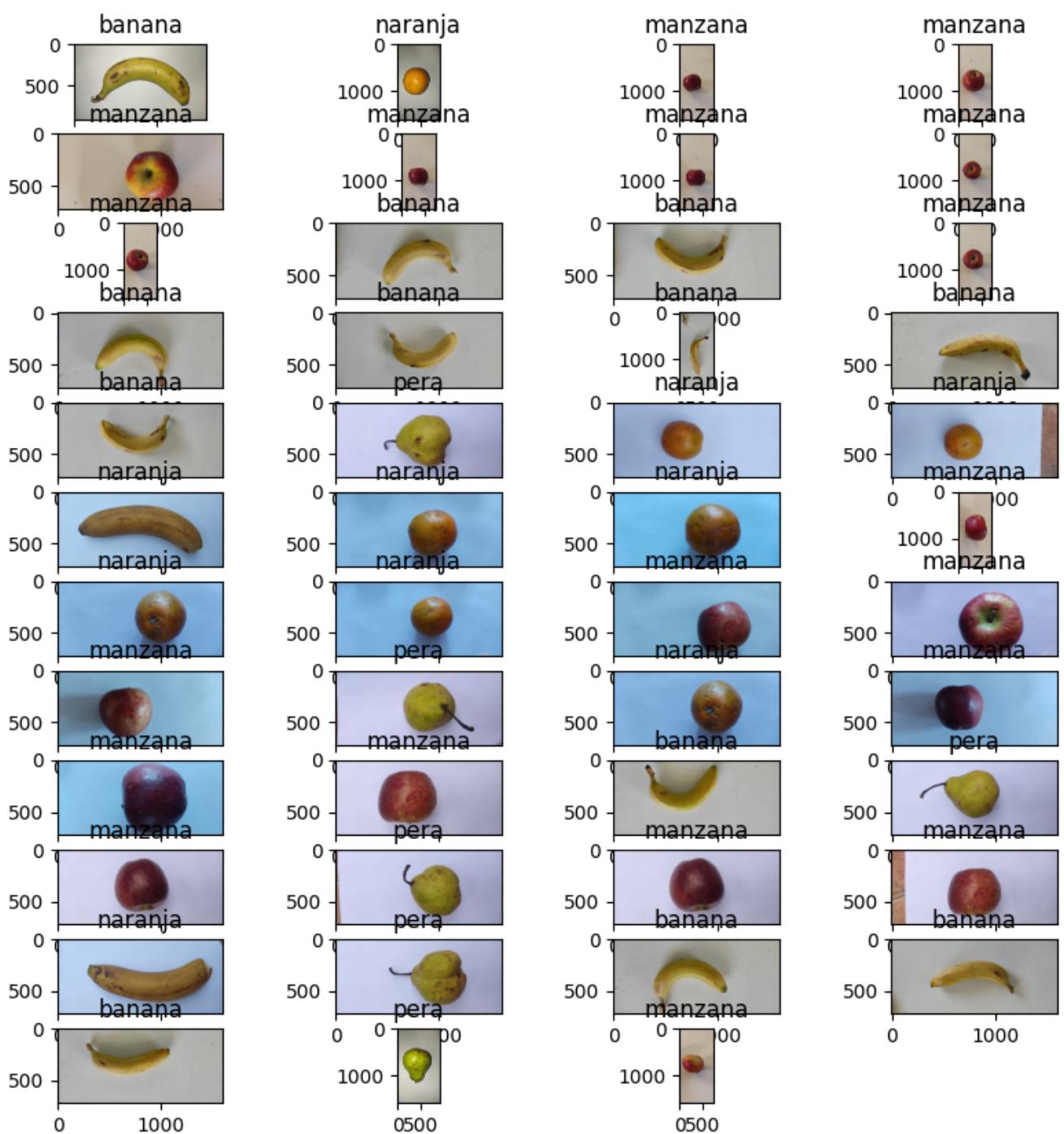
## REPRESENTACIÓN DE LAS IMÁGENES CON SUS ETIQUETAS

```
In [ ]: prediction = dict(zip(image_features.keys(), prediction))
total       = len(prediction.keys())
cols        = 4
rows        = total//cols
if total%cols != 0:
    rows += 1

i = 0
plt.figure(figsize = (10, 10))
for basename, label in prediction.items():

    file = os.path.join(original_path, basename)

    plt.subplot(rows, cols, i + 1)
    plt.imshow(cv2.cvtColor(cv2.imread(file), cv2.COLOR_BGR2RGB))
    plt.title(label)
    i += 1
plt.show()
```



## REPRESENTACIÓN DE LOS PUNTOS EN EL ESPACIO

Armamos los *clusters* del entrenamiento

```
In [ ]: points_matrix = np.vstack(list(training_features.values()))
training_points = dict.fromkeys(set(training_labels.values()))

for point, label in zip(points_matrix, training_labels.values()):
    if training_points[label] is None:
        training_points[label] = []
    training_points[label].append(point)

points = dict()
for fruit, cluster in training_points.items():
    points[fruit] = np.vstack(cluster)
```

Incorporamos los puntos de las frutas de validación al diccionario

```
In [ ]: points['validation'] = np.vstack(list(image_features.values()))
```

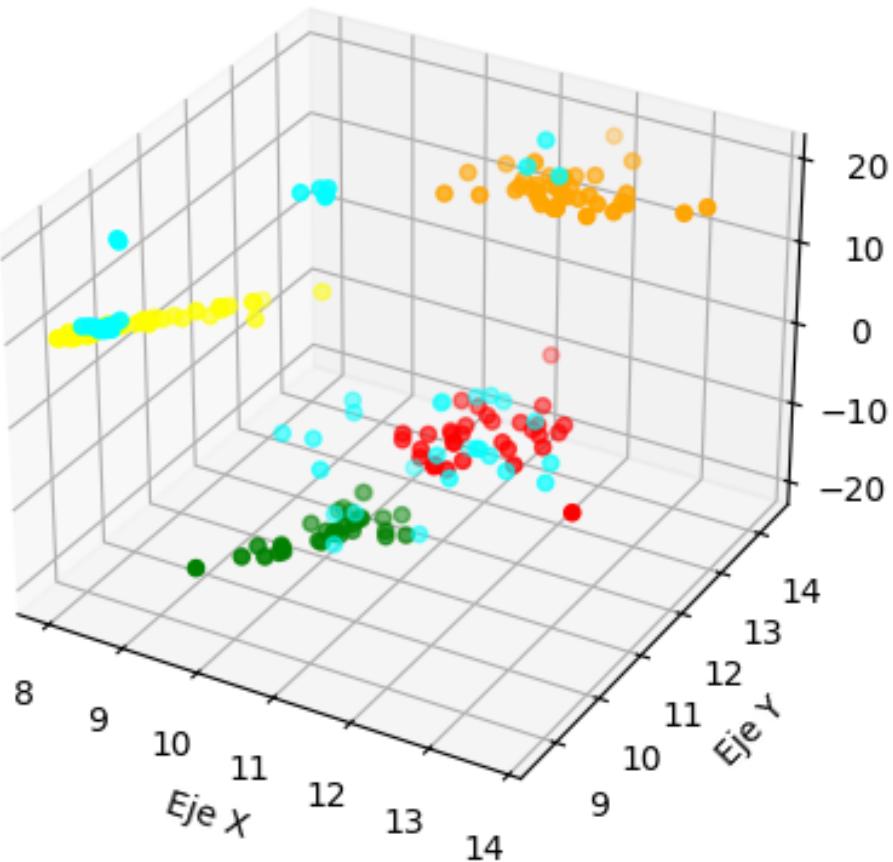
```
In [ ]: #3d
fig = plt.figure()
ax = fig.add_subplot(111, projection ='3d')

color_values = []
for key in points:
    if key == 'naranja':
        color_values.append('orange')
    elif key == 'banana':
        color_values.append('yellow')
    elif key == 'pera':
        color_values.append('green')
    elif key == 'manzana':
        color_values.append('red')
    else:
        color_values.append('cyan')

colors = dict(zip(points.keys(), color_values))

for key, cluster in points.items():
    ax.scatter(cluster[:, 0], cluster[:, 1], cluster[:, 2], c=colors[key], s=100)

ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')
plt.show()
```



## IMPORTS

```
In [ ]: import os
import math
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import joblib
from scipy.signal import butter, lfilter
import soundfile as sf
```

## VARIABLES GLOBALES

```
In [ ]: fruit_types      = ['pera', 'banana', 'manzana', 'naranja']
audios           = {fruit: [] for fruit in fruit_types}
training_path    = '../../../dataset/audios/training'
original_path    = os.path.join(training_path, 'original')
processed_path   = os.path.join(training_path, 'processed')
model_file       = 'model.pkl'
model            = dict.fromkeys(['pca', 'features', 'scaler'])
```

## DICCIONARIO DE AUDIOS ORIGINALES

```
In [ ]: original = {fruit: [] for fruit in fruit_types}
for dirname, _, filenames in os.walk(original_path):
    subdir = os.path.basename(dirname)
    if subdir in fruit_types:
        original[subdir].extend([os.path.join(dirname, filename) for filen
```

## DICCIONARIO DE AUDIOS PROCESADOS

```
In [ ]: processed = {fruit: [] for fruit in fruit_types}
for dirname, _, filenames in os.walk(processed_path):
    subdir = os.path.basename(dirname)
    if subdir in fruit_types:
        processed[subdir].extend([os.path.join(dirname, filename) for file
```

## PARAMETROS DEL AUDIO

```
In [ ]: FRAME_SIZE = 512# In the documentation says it's convenient for speech.C
HOP_SIZE     = int(FRAME_SIZE/2)
```

## FUNCIONES GENERALES DE AUDIO

```
In [ ]: def load_audio(audiofile):
    test_audio, sr = librosa.load(audiofile, sr = None)
    duration = librosa.get_duration(filename=audiofile, sr=sr)
    return test_audio, sr, duration
```

## FILTERS

```
In [ ]: def band_pass_filter(signal, sr, low_cutoff, high_cutoff):
    b, a = butter(N=3, Wn = [low_cutoff, high_cutoff], btype='band', fs=sr)
    return lfilter(b, a, signal)
```

## PRROCESSING OF THE AUDIO FILES FUNCTIONS

```
In [ ]: def spectral_flux(signal):

    # Calcular el espectrograma de magnitudes
    spectrogram = np.abs(librosa.stft(signal, n_fft = FRAME_SIZE, hop_length = HOP_SIZE))

    # Calcular el flujo espectral
    spectral_flux_values = np.sum(np.diff(spectrogram, axis=1)**2, axis=0)

    return spectral_flux_values
```

```
In [ ]: def process(audio_in, audio_out, rms_umbral = 0.043, flux_umbral = 0.096):
    signal, sr, _ = load_audio(audio_in)

    rms = librosa.feature.rms(y = signal, frame_length = FRAME_SIZE, hop_length = HOP_SIZE)
    rms /= np.max(np.abs(rms))
    trms = librosa.times_like(rms, sr = sr, hop_length = HOP_SIZE, n_fft = N_FFT)
    trms /= trms[-1]

    flux = spectral_flux(signal)
    flux /= np.max(np.abs(flux))
    fluxframes = range(len(flux))
    tflux = librosa.frames_to_time(fluxframes, hop_length=HOP_SIZE, n_fft = N_FFT)
    tflux /= tflux[-1]

    left_index = np.argmax(np.abs(flux) > flux_umbral)
    right_index = len(flux) - 1 - np.argmax(np.abs(np.flip(flux))) > flux_umbral

    tsignal = librosa.times_like(signal, sr = sr, hop_length=HOP_SIZE, n_fft = N_FFT)
    tsignal /= tsignal[-1]

    flag      = False
    pad_left = 0
    pad_right = 0
    flag_left = False
    flag_right = False

    while not flag:
        if rms[0, left_index] > rms_umbral:
            if left_index > pad_left + 15:
```

```

        rms_left = left_index - np.argmax(np.flip(np.abs(rms[0, :1
        if rms_left <= 0:
            rms_left = left_index
            flag_left = True
        else:
            pad_left += 15
            left_index = pad_left + np.argmax(np.abs(flux[pad_left:]))
    else:
        rms_left = left_index
        flag_left = True

    if rms[0, rigth_index] > rms_umbral:
        if rigth_index < (len(flux) - 1 - pad_rigth-15):
            rms_rigth = rigth_index + np.argmax(np.abs(rms[0, rigth_in
            if rms_rigth >= len(flux):
                rms_rigth = rigth_index
                flag_rigth = True
            else:
                pad_rigth += 15
                rigth_index = len(flux[:pad_rigth]) - 1 - np.argmax(np.fl
        else:
            rms_rigth = rigth_index
            flag_rigth = True

    flag = flag_left and flag_rigth

    left_index = min(left_index, rms_left)
    rigth_index = max(rigth_index, rms_rigth)

    mask = tsignal >= tflux[left_index]
    ttrimmed = tsignal[mask]
    trimed = signal[mask]
    mask = ttrimmed <= tflux[rigth_index]
    ttrimmed = ttrimmed[mask]
    trimed = trimed[mask]
    sf.write(audio_out, trimed, sr)

```

```

In [ ]: def process_audios(original:dict, processed:dict):
already_processed = []
for group in processed.values():
    already_processed.extend([os.path.basename(audio) for audio in gro

for fruit, audios in original.items():
    for audio in audios:
        file = os.path.basename(audio)
        if file in already_processed:
            pass
        else:
            audio_out = os.path.join(processed, f"{fruit}/{file}")
            process(audio, audio_out)
            processed[fruit].append(audio_out)

```

## PLOTTING

```
In [ ]: #3d
def plot_features3d(features):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    colors = dict(zip(fruit_types,['green','yellow','red','orange']))

    for fruit, points in features.items():
        ax.scatter(points[:, 0], points[:, 1], points[:, 2], c=colors[fruit])

    ax.set_xlabel('Eje X')
    ax.set_ylabel('Eje Y')
    ax.set_zlabel('Eje Z')
    plt.show()
```

## AUDIO PROCESSING

```
In [ ]: process_audios(original, processed)
```

## FEATURES EXTRACTION

*Features extraction functions*

```
In [ ]: def calculate_split_frequency_bin(split_frequency, sample_rate, num_frequency_bins):
    """Infer the frequency bin associated to a given split frequency."""

    frequency_range = sample_rate / 2
    frequency_delta_per_bin = frequency_range / num_frequency_bins
    split_frequency_bin = math.floor(split_frequency / frequency_delta_per_bin)
    return int(split_frequency_bin)
```

```
In [ ]: def band_energy_ratio(spectrogram, split_frequency, sample_rate):
    """Calculate band energy ratio with a given split frequency."""

    split_frequency_bin = calculate_split_frequency_bin(split_frequency, sample_rate)
    band_energy_ratio = []

    # calculate power spectrogram
    power_spectrogram = np.abs(spectrogram) ** 2
    power_spectrogram = power_spectrogram.T

    # calculate BER value for each frame
    for frame in power_spectrogram:
        sum_power_low_frequencies = frame[:split_frequency_bin].sum()
        sum_power_high_frequencies = frame[split_frequency_bin:].sum()
        band_energy_ratio_current_frame = sum_power_low_frequencies / (sum_power_low_frequencies + sum_power_high_frequencies)
        band_energy_ratio.append(band_energy_ratio_current_frame)

    return np.array(band_energy_ratio)
```

```
In [ ]: def rms(signal, frames, hop):
    return librosa.feature.rms(y=signal, frame_length = frames, hop_length = hop)
```

*function to get the features*

```
In [ ]: def get_features(signal, sr):
    feature = np.empty((1, 0))

    # BER
    spec = librosa.stft(signal, n_fft = FRAME_SIZE, hop_length = HOP_SIZE)
    # max
    split_frequency = 600
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.max(np.abs(BER))
    feature = np.append(feature, feat)
    # min
    # 1
    split_frequency = 1900
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.min(np.abs(BER))
    feature = np.append(feature, feat)
    # 2
    split_frequency = 5000
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.min(np.abs(BER))
    feature = np.append(feature, feat)
    # 3
    split_frequency = 9000
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.min(np.abs(BER))
    feature = np.append(feature, feat)
    # std
    # 1
    split_frequency = 8000
    BER = band_energy_ratio(spec, split_frequency, sr)
    BER /= np.max(np.abs(BER))
    feat = np.std(BER)/np.mean(np.abs(BER))
    feature = np.append(feature, feat)
    # 2
    split_frequency = 1000
    BER = band_energy_ratio(spec, split_frequency, sr)
    BER /= np.max(np.abs(BER))
    feat = np.std(BER)/np.mean(np.abs(BER))
    feature = np.append(feature, feat)

#ZCR
cutoff = 5000
cuton = 1000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
zcr /= np.max(np.abs(zcr))
# mean
feat = np.mean(zcr)
feature = np.append(feature, feat)
# maximum
cutoff = 10000
```

```

cuton = 10
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
feat = np.max(np.abs(zcr))
feature = np.append(feature, feat)
# std
cutoff = 10000
cuton = 20
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
feat = np.std(zcr)/np.mean(np.abs(zcr))
feature = np.append(feature, feat)
# mean Local
cutoff = 5000
cuton = 1000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
zcr /= np.max(np.abs(zcr))
feat = np.mean(zcr[((len(zcr)*3)//14 - 5) : ((len(zcr)*3)//14 + 5)])
feature = np.append(feature, feat)
# Local max
cutoff = 10000
cuton = 10
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
feat = np.max(zcr[((len(zcr)*3)//4 - 10) : ((len(zcr)*3)//4 + 10)])
feature = np.append(feature, feat)

# Roll off
cuton = 100
cutoff = 8500
filtered = band_pass_filter(signal, sr, cuton, cutoff)
roll_off = librosa.feature.spectral_rolloff(y=filtered, sr=sr, n_fft=F
roll_off /= np.max(np.abs(roll_off))
# mean
feat = np.mean(np.abs(roll_off))
feature = np.append(feature, feat)
# max
cuton = 100
cutoff = 8500
filtered = band_pass_filter(signal, sr, cuton, cutoff)
roll_off = librosa.feature.spectral_rolloff(y=filtered, sr=sr, n_fft=F
feat = np.max(np.abs(roll_off))
feature = np.append(feature, feat)
# std
cutoff = 8500
cuton = 50
filtered = band_pass_filter(signal, sr, cuton, cutoff)
roll_off = librosa.feature.spectral_rolloff(y=filtered, sr=sr, n_fft=F
roll_off /= np.max(np.abs(roll_off))
feat = np.std(np.abs(roll_off))/np.mean(np.abs(roll_off))
feature = np.append(feature, feat)

```

#MFCCS

```

n_mfcc = 4
# 1
cuton = 500
cutoff = 5000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
mfccs = librosa.feature.mfcc(y = signal, sr=sr, n_mfcc = n_mfcc, n_fft=1024)
feat = np.max(mfccs, axis = 1)
feature = np.append(feature, feat[3])

# 2
cuton = 10
cutoff = 8000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
mfccs = librosa.feature.mfcc(y = filtered, sr=sr, n_mfcc = n_mfcc, n_fft=1024)
mfccs /= np.max(np.abs(mfccs), axis = 1, keepdims=True)
feat = np.std(np.abs(mfccs), axis = 1)/np.mean(np.abs(mfccs), axis = 1)
feature = np.append(feature, feat[3])

# 3
cuton = 10
cutoff = 8000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
mfccs = librosa.feature.mfcc(y = filtered, sr=sr, n_mfcc = n_mfcc, n_fft=1024)
mfccs /= np.max(np.abs(mfccs), axis = 1, keepdims=True)
mfccs = mfccs[:, ((mfccs.shape[1]*4) // 5 - 10):((mfccs.shape[1]*4) // 5 + 10)]
feat = np.std(np.abs(mfccs), axis=1) / np.mean(np.abs(mfccs), axis=1)
feature = np.append(feature, feat[1])

#envelope
env = rms(signal, FRAME_SIZE, HOP_SIZE)
env = env.reshape(-1,1)
selected = np.linspace(0, len(env) - 1, 30, dtype=int)
env = env[selected]
feat = env[11]
feature = np.append(feature, feat)
feat = env[12]
feature = np.append(feature, feat)

return feature

```

## EXTRACTION OF FEATURES FROM PROCESSED AUDIOS

```

In [ ]: def extract_features(processed:dict):
    features = dict.fromkeys(fruit_types)
    for fruit, audios in processed.items():
        features[fruit] = None

        for audio in audios:
            # Load the audio signal
            signal, sr, _ = load_audio(audio)
            feature = get_features(signal, sr)

            if features[fruit] is not None:

```

```

        features[fruit] = np.vstack([features[fruit], feature])
    else:
        features[fruit] = feature
return features

```

## TRAINING AUDIOS FEATURES EXTRACTION

In [ ]: `features = extract_features(processed)`

```

C:\Users\Juan\AppData\Local\Temp\ipykernel_3228\1587896153.py:3: FutureWarning: get_duration() keyword argument 'filename' has been renamed to 'path' in version 0.10.0.
    This alias will be removed in version 1.0.
duration = librosa.get_duration(filename=audiofile, sr=sr)
c:\Users\Juan\AppData\Local\Programs\Python\Python311\Lib\site-packages\lib
rosa\feature\spectral.py:2143: UserWarning: Empty filters detected in mel f
requency basis. Some channels will produce empty responses. Try increasing
your sampling rate (and fmax) or reducing n_mels.
mel_basis = filters.mel(sr=sr, n_fft=n_fft, **kwargs)

```

## PCA

In [ ]: `#PCA and dump`

```

whole           = np.concatenate(list(features.values()), axis=0)

#Aplicar PCA para obtener dos componentes principales
pca            = PCA(n_components = 3)
scaler          = StandardScaler()
whole_scaled   = scaler.fit_transform(whole)
reduced_features = pca.fit_transform(whole_scaled)

```

## REDUCED MODEL

In [ ]: `#Paso 3: Crear un diccionario con las matrices reducidas`

```

reduced = {}
start_idx = 0

for fruit, matrix in features.items():
    num_rows = matrix.shape[0]
    reduced[fruit] = reduced_features[start_idx:start_idx + num_rows, :]
    start_idx += num_rows

```

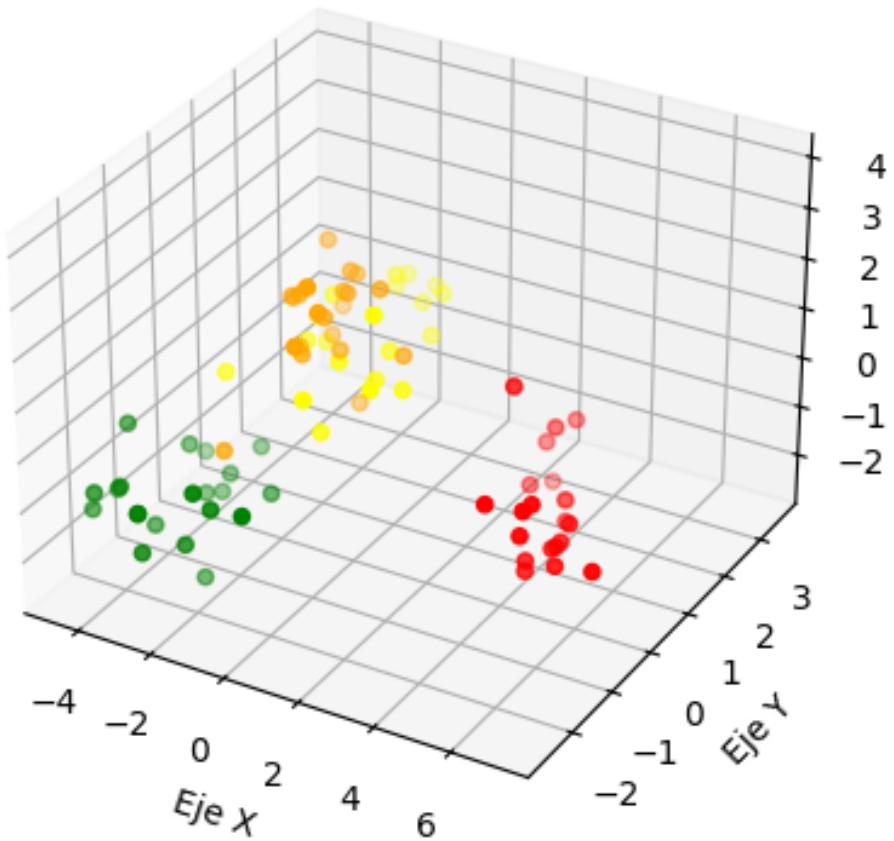
## DUMPING TO FILE

In [ ]: `model['pca'] = pca`  
`model['features'] = reduced`  
`model['scaler'] = scaler`  
`joblib.dump(model, model_file)`

Out[ ]: `['model.pkl']`

## PLOTTING

```
In [ ]: plot_features3d(reduced)
```



## IMPORTS

```
In [ ]: import os
import math
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from scipy.spatial.distance import cdist
import joblib
import sounddevice as sd
from scipy.signal import butter, lfilter
import soundfile as sf
```

## RUTAS Y TIPOS DE FRUTAS

```
In [ ]: fruit_types      = ['pera', 'banana', 'manzana', 'naranja']
dataset_path       = '../dataset/audios/validation'
original_path     = os.path.join(dataset_path, 'original')
processed_path    = os.path.join(dataset_path, 'processed')
model_file        = 'model.pkl'
model             = dict.fromkeys(['pca', 'features', 'scaler'])
```

## ORIGINAL TESTS

```
In [ ]: original = []
original.extend([os.path.join(original_path, filename) for filename in os.
```

## PROCESSED TESTS DICT

```
In [ ]: processed = []
processed.extend([os.path.join(processed_path, filename) for filename in o
```

## PARAMETROS DEL AUDIO

```
In [ ]: FRAME_SIZE = 512# In the documentation says it's convenient for speech.
HOP_SIZE   = int(FRAME_SIZE/2)
```

## FUNCIONES GENERALES DE AUDIO

```
In [ ]: def load_audio(audiofile):
    test_audio, sr = librosa.load(audiofile, sr = None)
    duration = librosa.get_duration(filename=audiofile, sr=sr)
    return test_audio, sr, duration
```

## FILTERS

```
In [ ]: def band_pass_filter(signal, sr, low_cutoff, high_cutoff):
    b, a = butter(N=3, Wn = [low_cutoff, high_cutoff], btype='band', fs=sr
    return lfilter(b, a, signal)
```

## PROCESSING OF THE AUDIO FILES FUNCTIONS

### File naming

```
In [ ]: def get_name(original:list):
    return os.path.join(original_path,"validation" + f"{len(original) + 1}")
```

### Processing

```
In [ ]: def spectral_flux(signal):

    # Calcular el espectrograma de magnitudes
    spectrogram = np.abs(librosa.stft(signal, n_fft = FRAME_SIZE, hop_length = HOP_SIZE))

    # Calcular el flujo espectral
    spectral_flux_values = np.sum(np.diff(spectrogram, axis=1)**2, axis=0)

    return spectral_flux_values
```

```
In [ ]: def process(audio_in, audio_out, rms_umbral = 0.043, flux_umbral = 0.096):
    signal, sr, _ = load_audio(audio_in)

    rms = librosa.feature.rms(y = signal, frame_length = FRAME_SIZE, hop_length = HOP_SIZE)
    rms /= np.max(np.abs(rms))
    trms = librosa.times_like(rms, sr = sr, hop_length = HOP_SIZE, n_fft = n_fft)
    trms /= trms[-1]

    flux = spectral_flux(signal)
    flux /= np.max(np.abs(flux))
    fluxframes = range(len(flux))
    tflux = librosa.frames_to_time(fluxframes, hop_length=HOP_SIZE, n_fft = n_fft)
    tflux /= tflux[-1]

    left_index = np.argmax(np.abs(flux) > flux_umbral)
    right_index = len(flux) - 1 - np.argmax(np.abs(np.flip(flux))) > flux_umbral

    tsignal = librosa.times_like(signal, sr = sr, hop_length=HOP_SIZE, n_fft = n_fft)
    tsignal /= tsignal[-1]

    flag = False
    pad_left = 0
    pad_right = 0
    flag_left = False
    flag_right = False

    while not flag:
        if rms[0, left_index] > rms_umbral:
```

```

    if left_index > pad_left + 15:
        rms_left = left_index - np.argmax(np.flip(np.abs(rms[0, :1
        if rms_left <= 0:
            rms_left = left_index
            flag_left = True
        else:
            pad_left += 15
            left_index = pad_left + np.argmax(np.abs(flux[pad_left:]))
    else:
        rms_left = left_index
        flag_left = True

    if rms[0, rigth_index] > rms_umbral:
        if rigth_index < (len(flux) - 1 - pad_rigth-15):
            rms_rigth = rigth_index + np.argmax(np.abs(rms[0, rigth_in
            if rms_rigth >= len(flux):
                rms_rigth = rigth_index
                flag_rigth = True
            else:
                pad_rigth += 15
                rigth_index = len(flux[:pad_rigth]) - 1 - np.argmax(np.fl
        else:
            rms_rigth = rigth_index
            flag_rigth = True

    flag = flag_left and flag_rigth

    left_index = min(left_index, rms_left)
    rigth_index = max(rigth_index, rms_rigth)
    mask = tsignal >= tflux[left_index]
    ttrimmed = tsignal[mask]
    trimed = signal[mask]
    mask = ttrimmed <= tflux[rigth_index]
    ttrimmed = ttrimmed[mask]
    trimed = trimed[mask]
    sf.write(audio_out, trimed, sr)

```

## KNN

```

In [ ]: def knn(training, test, k_n):
    X          = np.concatenate([v for v in training.values()], axis = 0)
    y          = np.concatenate([[k] * v.shape[0] for k, v in training.it
    dist       = cdist(test, X)
    sorted_ind = np.argsort(dist, axis = 1)
    sorted_k   = sorted_ind[:, 0:k_n]
    predicted  = []

    for row in sorted_k:
        labels    = list(y[row])
        prediction = max(set(labels), key = labels.count)
        predicted.append(prediction)
    return predicted

```

## PLOTTING

```
In [ ]: #3d
def plot_features3d(features):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    colors = dict(zip(fruit_types,['green','yellow','red','orange']))

    for fruit, points in features.items():
        ax.scatter(points[:, 0], points[:, 1], points[:, 2], c=colors[fruit])

    ax.set_xlabel('Eje X')
    ax.set_ylabel('Eje Y')
    ax.set_zlabel('Eje Z')
    plt.show()
```

```
In [ ]: #3d
def plot_features3d_extra(features):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    colors = dict(zip(features.keys(),['green','yellow','red','orange','cyan']))

    for fruit, points in features.items():
        ax.scatter(points[:, 0], points[:, 1], points[:, 2], c=colors[fruit])
        if fruit == 'validation':
            for i, point in enumerate(points):
                ax.text(point[0], point[1], point[2], f"{i}", color='red', size=10)
    ax.set_xlabel('Eje X')
    ax.set_ylabel('Eje Y')
    ax.set_zlabel('Eje Z')
    plt.show()
```

## FEATURES EXTRACTION

*Features extraction functions*

```
In [ ]: def calculate_split_frequency_bin(split_frequency, sample_rate, num_frequency_bins):
    """Infer the frequency bin associated to a given split frequency."""

    frequency_range = sample_rate / 2
    frequency_delta_per_bin = frequency_range / num_frequency_bins
    split_frequency_bin = math.floor(split_frequency / frequency_delta_per_bin)
    return int(split_frequency_bin)
```

```
In [ ]: def band_energy_ratio(spectrogram, split_frequency, sample_rate):
    """Calculate band energy ratio with a given split frequency."""

    split_frequency_bin = calculate_split_frequency_bin(split_frequency, sample_rate)
    band_energy_ratio = []

    # calculate power spectrogram
    power_spectrogram = np.abs(spectrogram) ** 2
```

```

power_spectrogram = power_spectrogram.T

# calculate BER value for each frame
for frame in power_spectrogram:
    sum_power_low_frequencies = frame[:split_frequency_bin].sum()
    sum_power_high_frequencies = frame[split_frequency_bin:].sum()
    band_energy_ratio_current_frame = sum_power_low_frequencies / (sum_power_low_frequencies + sum_power_high_frequencies)
    band_energy_ratio.append(band_energy_ratio_current_frame)

return np.array(band_energy_ratio)

```

```
In [ ]: def rms(signal, frames, hop):
    return librosa.feature.rms(y=signal, frame_length = frames, hop_length = hop)
```

*function to get the features*

```

In [ ]: def get_features(signal, sr):
    feature = np.empty((1, 0))

    # BER
    spec = librosa.stft(signal, n_fft = FRAME_SIZE, hop_length = HOP_SIZE)
    # max
    split_frequency = 600
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.max(np.abs(BER))
    feature = np.append(feature, feat)
    # min
    # 1
    split_frequency = 1900
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.min(np.abs(BER))
    feature = np.append(feature, feat)
    # 2
    split_frequency = 5000
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.min(np.abs(BER))
    feature = np.append(feature, feat)
    # 3
    split_frequency = 9000
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.min(np.abs(BER))
    feature = np.append(feature, feat)
    # std
    # 1
    split_frequency = 8000
    BER = band_energy_ratio(spec, split_frequency, sr)
    BER /= np.max(np.abs(BER))
    feat = np.std(BER)/np.mean(np.abs(BER))
    feature = np.append(feature, feat)
    # 2
    split_frequency = 1000
    BER = band_energy_ratio(spec, split_frequency, sr)
    BER /= np.max(np.abs(BER))

```

```

feat = np.std(BER)/np.mean(np.abs(BER))
feature = np.append(feature, feat)

#ZCR
cutoff = 5000
cuton = 1000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
zcr /= np.max(np.abs(zcr))
# mean
feat = np.mean(zcr)
feature = np.append(feature, feat)
# maximum
cutoff = 10000
cuton = 10
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
feat = np.max(np.abs(zcr))
feature = np.append(feature, feat)
# std
cutoff = 10000
cuton = 20
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
feat = np.std(zcr)/np.mean(np.abs(zcr))
feature = np.append(feature, feat)
# mean Local
cutoff = 5000
cuton = 1000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
zcr /= np.max(np.abs(zcr))
feat = np.mean(zcr[((len(zcr)*3)//14 - 5) : ((len(zcr)*3)//14 + 5)])
feature = np.append(feature, feat)
# Local max
cutoff = 10000
cuton = 10
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered, frame_length=FRAME_
feat = np.max(zcr[((len(zcr)*3)//4 - 10) : ((len(zcr)*3)//4 + 10)])
feature = np.append(feature, feat)

# Roll off
cuton = 100
cutoff = 8500
filtered = band_pass_filter(signal, sr, cuton, cutoff)
roll_off = librosa.feature.spectral_rolloff(y=filtered, sr=sr, n_fft=F
roll_off /= np.max(np.abs(roll_off))
# mean
feat = np.mean(np.abs(roll_off))
feature = np.append(feature, feat)
# max
cuton = 100
cutoff = 8500

```

```

filtered = band_pass_filter(signal, sr, cuton, cutoff)
roll_off = librosa.feature.spectral_rolloff(y=filtered, sr=sr, n_fft=F
feat = np.max(np.abs(roll_off))
feature = np.append(feature, feat)
# std
cutoff = 8500
cuton = 50
filtered = band_pass_filter(signal, sr, cuton, cutoff)
roll_off = librosa.feature.spectral_rolloff(y=filtered, sr=sr, n_fft=F
roll_off /= np.max(np.abs(roll_off))
feat = np.std(np.abs(roll_off))/np.mean(np.abs(roll_off))
feature = np.append(feature, feat)

#MFCCS
n_mfcc = 4
# 1
cuton = 500
cutoff = 5000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
mfccs = librosa.feature.mfcc(y = signal, sr=sr, n_mfcc = n_mfcc, n_fft=
feat = np.max(mfccs, axis = 1)
feature = np.append(feature, feat[3])

# 2
cuton = 10
cutoff = 8000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
mfccs = librosa.feature.mfcc(y = filtered, sr=sr, n_mfcc = n_mfcc, n_f
mfccs /= np.max(np.abs(mfccs), axis = 1, keepdims=True)
feat = np.std(np.abs(mfccs), axis = 1)/np.mean(np.abs(mfccs), axis = 1
feature = np.append(feature, feat[3])

# 3
cuton = 10
cutoff = 8000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
mfccs = librosa.feature.mfcc(y = filtered, sr=sr, n_mfcc = n_mfcc, n_f
mfccs /= np.max(np.abs(mfccs), axis = 1, keepdims=True)
mfccs = mfccs[:, ((mfccs.shape[1]*4) // 5 - 10):((mfccs.shape[1]*4) // 5 + 10)]
feat = np.std(np.abs(mfccs), axis=1) / np.mean(np.abs(mfccs), axis=1)
feature = np.append(feature, feat[1])

#envelope
env = rms(signal, FRAME_SIZE, HOP_SIZE)
env = env.reshape(-1,)
selected = np.linspace(0, len(env) - 1, 30, dtype=int)
env = env[selected]
feat = env[11]
feature = np.append(feature, feat)
feat = env[12]
feature = np.append(feature, feat)

return feature

```

## AUDIO RECORDING

```
In [ ]: '''duration = 3      # Duración de la grabación en segundos
fs       = 48000   # Frecuencia de muestreo en Hz

print("Grabando...")
data = sd.rec(int(duration * fs), samplerate = fs, channels = 1, dtype = 'int16')
sd.wait()
print("Grabación completa.")

file = get_name(original)
sf.write(file, data, fs)
original.append(file)'''
```

```
Out[ ]: 'duration = 3      # Duración de la grabación en segundos\nfs       = 48000   # Frecuencia de muestreo en Hz\n\nprint("Grabando...")\nadata = sd.rec(\n    int(duration * fs), samplerate = fs, channels = 1, dtype = 'int16')\nsd.wait()\nprint("Grabación completa.")\n\nfile = get_name(original)\nsf.write(file, data, fs)\noriginal.append(file)'
```

## PROCESSING

```
In [ ]: already_processed = [os.path.basename(audio) for audio in processed]
for audio in original:
    basename = os.path.basename(audio)
    if basename in already_processed:
        pass
    else:
        audio_out = os.path.join(processed_path, basename)
        process(audio, audio_out)
        processed.append(audio_out)
```

## FEATURE EXTRACTION

```
In [ ]: validation_features = None
for audio in processed:
    signal, sr, _ = load_audio(audio)
    feature = get_features(signal, sr)

    if validation_features is not None:
        validation_features = np.vstack([validation_features, feature])
    else:
        validation_features = feature.reshape(1, -1)
```

```
C:\Users\Juan\AppData\Local\Temp\ipykernel_1852\1587896153.py:3: FutureWarning: get_duration() keyword argument 'filename' has been renamed to 'path' in version 0.10.0.
    This alias will be removed in version 1.0.
    duration = librosa.get_duration(filename=audiofile, sr=sr)
c:\Users\Juan\AppData\Local\Programs\Python\Python311\Lib\site-packages\lib
rosa\feature\spectral.py:2143: UserWarning: Empty filters detected in mel f
requency basis. Some channels will produce empty responses. Try increasing
your sampling rate (and fmax) or reducing n_mels.
    mel_basis = filters.mel(sr=sr, n_fft=n_fft, **kwargs)
```

## LOAD THE REDUCED MODEL

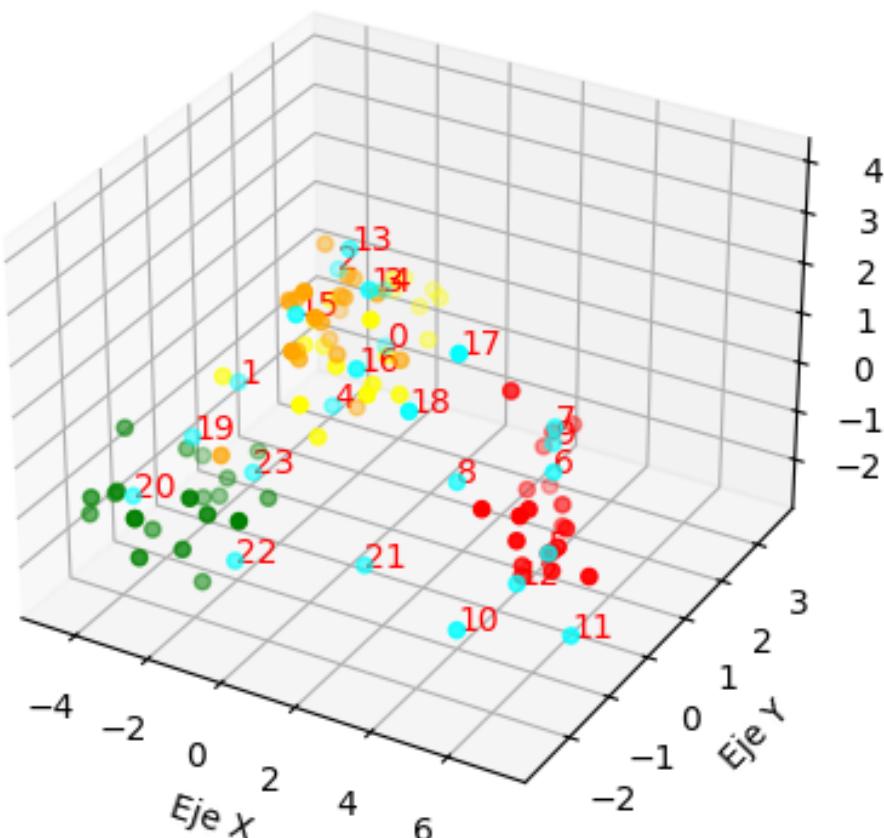
```
In [ ]: model      = joblib.load(model_file)
reduced_dict = model['features']
pca         = model['pca']
scaler       = model['scaler']
```

## TRANSFORM

```
In [ ]: scaled_validation_features = scaler.transform(validation_features)
reduced_validation      = pca.transform(scaled_validation_features)
```

## EXTENSION OF THE FEATURES DICT

```
In [ ]: extra           = reduced.copy()
extra['validation'] = reduced_validation
plot_features3d_extra(extra)
```



## PREDICTION

```
In [ ]: from prettytable import PrettyTable

prediction = knn(reduced, reduced_validation, 3)
aciertos = []
audios = []
N_aciertos = 0

for i, audio in enumerate(processed):
    audios.append(os.path.basename(audio))
    if prediction[i] in os.path.basename(audio):
        aciertos.append('acerto')
        N_aciertos += 1
    else:
        aciertos.append('falla')
precision = N_aciertos*100/len(aciertos)

table = PrettyTable()
table.field_names = ['audios'] + audios
table.add_row(['prediction'] + prediction)
table.add_row(['results'] + aciertos)

print(table)
print(f"Se acertaron: {N_aciertos}/{len(prediction)}")
print(f"El porcentaje de aciertos es: {precision}")
```

Se acertaron: 24/24

El porcentaje de aciertos es: 100.0

```

# IMAGEN
#-----IMPORTS-----
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import joblib
from scipy.spatial.distance import cdist
import math
import librosa
import librosa.display
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import sounddevice as sd
from scipy.signal import butter, lfilter
import soundfile as sf
import warnings

warnings.filterwarnings("ignore", message="Empty filters detected in mel
frequency basis", category=UserWarning)
#-----PATHS-----
-----
image_path      = './dataset/images'
shelves_path   = os.path.join(image_path, 'test')
shelf_names     = ['shelf1', 'shelf2', 'shelf3', 'shelf4']
training_data   = './implementation/images/kmeans/training_data.pkl'
#-----DICCCIONARIO CON IMAGENES-----
-----
shelf_files = dict()
for shelf in shelf_names:
    shelf_dir      = os.path.join(shelves_path, shelf)
    shelf_original = os.path.join(shelf_dir, 'original')
    image_files    = os.listdir(shelf_original)

    shelf_files[shelf] = os.path.join(shelf_original, image_files[0])
#-----PROCESAMIENTO DE IMAGENES-----
-----
def get_light_background(mask, f = 20, p = 0.75):
    height, width = mask.shape
    cluster_size   = min([height, width])//f
    cluster        = np.ones((cluster_size, cluster_size), np.uint8)

    # Corners
    corner1 = np.bitwise_and(cluster, mask[:cluster_size,
:cluster_size])
    corner2 = np.bitwise_and(cluster, mask[:cluster_size:, -
cluster_size:])
    corner3 = np.bitwise_and(cluster, mask[-cluster_size:, -
cluster_size])
    corner4 = np.bitwise_and(cluster, mask[-cluster_size:, -
cluster_size:])
    corners = [corner1, corner2, corner3, corner4]

```

```

# Sides
limitwl = (width - cluster_size)//2
limitw2 = (width + cluster_size)//2
limith1 = (height - cluster_size)//2
limith2 = (height + cluster_size)//2

    side1 = np.bitwise_and(cluster, mask[:cluster_size,
limitwl:limitw2])
    side2 = np.bitwise_and(cluster, mask[limith1:limith2,
:cluster_size])
    side3 = np.bitwise_and(cluster, mask[limith1:limith2, -
cluster_size:])
    side4 = np.bitwise_and(cluster, mask[-cluster_size:, -
limitwl:limitw2])
    sides = [side1, side2, side3, side4]

# Determining the type of background
edges = corners + sides
light_background = sum(np.count_nonzero(edge) for edge in edges) >
p*8*(cluster_size**2)

# Inverting if dark background
if light_background:
    return np.bitwise_not(mask)
return mask
-----OBTENCIÓN DE MÁSCARAS-----
os.system('cls')
print('Procesando las imágenes...')
for shelf, file in shelf_files.items():
    # BGR image
    image = cv2.imread(file)

    # Dimensions
    height, width, _ = image.shape

    # Pixel data vector
    data_vector = np.zeros((height * width, 4))

    # Obtain matrices of the color space
    rgb_matrix = image.reshape((-1, 3))
    hsv_matrix = cv2.cvtColor(image, cv2.COLOR_BGR2HSV).reshape((-1, 3))
    lab_matrix = cv2.cvtColor(image, cv2.COLOR_BGR2LAB).reshape((-1, 3))

    # Assign to the data matrix
    # We keep the G, S, A and B channels
    data_vector[:, 0] = rgb_matrix[:, 2]
    data_vector[:, 1] = hsv_matrix[:, 1]
    data_vector[:, 2:] = lab_matrix[:, 1:]

    # Segmentamos la imagen con los vectores obtenidos por cada pixel

```

```

kmeans = KMeans(n_clusters = 2, n_init = 10, random_state=42) # 2
Clusters. Background and fruit
kmeans.fit(data_vector)

# Get clusters labels
labels = kmeans.labels_

# kmeans_mask
kmeans_mask = labels.reshape(height, width)
kmeans_mask = kmeans_mask.astype(np.uint8) * 255

# Determinación del tipo de fondo de la máscara
kmeans_mask = get_light_background(kmeans_mask)

# Erosion y dilataciòn sobre la màscara
erosion_size      = min([height, width])//200
dilatacion_size   = min([height, width])//80
kernel_erosion    = np.ones((erosion_size,erosion_size), np.uint8)
eroded            = cv2.erode(kmeans_mask, kernel_erosion, iterations
= 1)
kernel_dilatacion = np.ones((dilatacion_size,dilatacion_size),
np.uint8)
kmeans_mask       = cv2.dilate(eroded, kernel_dilatacion, iterations
= 2)

# Encontrar contornos
kmeans_cnt, _ = cv2.findContours(kmeans_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
kmeans_cnt      = max(kmeans_cnt, key = cv2.contourArea)

# Contorno aproximado
epsilon          = 0.001 * cv2.arcLength(kmeans_cnt, True)
kmeans_cnt       = cv2.approxPolyDP(kmeans_cnt, epsilon, True)
kmeans_cnt       = (kmeans_cnt,)

# Template
tkmeans         = np.zeros((height, width), dtype=np.uint8)

# Dibujar
cv2.drawContours(tkmeans, kmeans_cnt, -1, 255, thickness =
cv2.FILLED)

# Guardar mascara
cv2.imwrite(os.path.join(shelfs_path,
f'{shelf}/processed/{os.path.basename(file)}'), tkmeans)
#-----DICCCIONARIO DE MASCARAS-----
-----
shelf_masks = dict()
for shelf in shelf_names:
    shelf_dir     = os.path.join(shelfs_path, shelf)
    shelf_mask    = os.path.join(shelf_dir, 'processed')
    mask_files   = os.listdir(shelf_mask)

```

```

shelf_masks[shelf] = os.path.join(shelf_mask, mask_files[0])
#-----RANGOS DE COLOR-----
-----
lower_red_2 = np.array([170, 60, 60])
upper_red_2 = np.array([179, 255, 255])

lower_red_1 = np.array([0, 60, 60])
upper_red_1 = np.array([8, 255, 255])

lower_orange = np.array([8, 120, 80])
upper_orange = np.array([21, 255, 255])

lower_yellow = np.array([21, 50, 80])
upper_yellow = np.array([25, 255, 255])

lower_green = np.array([25, 40, 40])
upper_green = np.array([100, 255, 255])
#-----EXTRACCIÓN DE CARACTERÍSTICAS IMAGENES-----
-----
print('Extrayendo características de las imágenes...')
conversion_color = {'V' : -20, 'R' : -10, 'A' : 10, 'N' : 20}
image_features = dict.fromkeys(shelf_names)

for shelf, image_file, mask_file in zip(shelf_files.keys(),
                                         shelf_files.values(), shelf_masks.values()):
    # Leer la imagen y la máscara
    image = cv2.imread(image_file)
    mask = cv2.imread(mask_file, cv2.IMREAD_GRAYSCALE)

    # Convertir la imagen de BGR a HSV
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Aplicar la máscara
    fruit = cv2.bitwise_and(hsv_image, hsv_image, mask=mask)

#-----Extracción de los momentos de Hu-----
-
# Encontrar el rectángulo delimitador de la fruta
(x, y, w, h) = cv2.boundingRect(mask)

# Recortar la imagen original para obtener solo la región de la fruta
trimed = fruit[y:y + h, x:x + w]

# Convertir la imagen a escala de grises si es necesario
trimed_gray = cv2.cvtColor(trimed, cv2.COLOR_BGR2GRAY)

# Calcular los momentos de la imagen
momentos = cv2.moments(trimed_gray)

# Calcular los momentos de Hu
momentos_hu = cv2.HuMoments(momentos)

```

```

# Aplicar logaritmo a los momentos de Hu para mejorar la escala
log_moments_hu = -np.sign(momentos_hu) *
np.log10(np.abs(momentos_hu))
moments = log_moments_hu.reshape(-1)

#-----Extracción de color-----
conteo = {
    'V' : np.sum(np.all(np.logical_and(lower_green <= fruit, fruit
<= upper_green), axis=-1)),
    'R1': np.sum(np.all(np.logical_and(lower_red_1 <= fruit, fruit
<= upper_red_1), axis=-1)),
    'R2': np.sum(np.all(np.logical_and(lower_red_2 <= fruit, fruit
<= upper_red_2), axis=-1)),
    'A' : np.sum(np.all(np.logical_and(lower_yellow <= fruit, fruit
<= upper_yellow), axis=-1)),
    'N' : np.sum(np.all(np.logical_and(lower_orange <= fruit, fruit
<= upper_orange), axis=-1))
}
conteo_por_rango = {
    'V': conteo['V'],
    'R': conteo['R1'] + conteo['R2'],
    'A': conteo['A'],
    'N': conteo['N']
}

sorted_conteo = sorted(conteo_por_rango.items(), key=lambda x: x[1],
reverse=True)

# Obtener el segundo elemento más grande
segundo_mas_grande = sorted_conteo[1]

# Obtener la etiqueta y el valor del segundo elemento más grande
etiqueta_segundo_mas_grande = segundo_mas_grande[0]
valor_segundo_mas_grande = segundo_mas_grande[1]

# Obtener la etiqueta basándose en el rango con el mayor conteo
etiqueta = max(conteo_por_rango, key = conteo_por_rango.get)

# Se usa el hecho de que a excepción de las manzanas, el resto de las
frutas tienen poco rojo
if (etiqueta_segundo_mas_grande == 'R') and (valor_segundo_mas_grande >
0.35*conteo_por_rango[etiqueta]):
    etiqueta = 'R'

color = conversion_color[etiqueta]

#-----Vector de características-----
image_features[shelf] = np.append(moments[2:4], color)

#-----RECUPERACIÓN CENTROIDES Y KNN-----
def knn(training, test, k_n):

```

```

X          = np.concatenate([v for v in training.values()], axis =
0)
y          = np.concatenate([[k] * v.shape[0] for k, v in
training.items()])
dist       = cdist(test, X)
sorted_ind = np.argsort(dist, axis = 1)
sorted_k   = sorted_ind[:, 0:k_n]
predicted  = []

for row in sorted_k:
    labels    = list(y[row])
    prediction = max(set(labels), key = labels.count)
    predicted.append(prediction)
return predicted

#-----DATOS DEL ENTRENAMIENTO-----
-----
data        = joblib.load(training_data)
centroids  = data['centroids']
prediction  = knn(centroids,
np.vstack(list(image_features.values())))
#-----CLASIFICACIÓN DE FRUTAS EN LOS ESTANTES-----
-----
shelves = dict(zip(image_features.keys(), prediction))

#-----RECONOCIMIENTO DE VOZ-----
-----
fruit_types = ['pera', 'banana', 'manzana', 'naranja']
dataset_path = './dataset/audios/test'
original_path = os.path.join(dataset_path, 'original')
processed_path = os.path.join(dataset_path, 'processed')
model_file   = './implementation/audio/knn/model.pkl'
model        = dict.fromkeys(['pca', 'features', 'scaler'])

#-----PARAMETROS DE AUDIO-----
-----
FRAME_SIZE = 512
HOP_SIZE   = int(FRAME_SIZE/2)
#-----GENERAL AUDIO FUNCTIONS-----
-----

def load_audio(audiofile):
    test_audio, sr = librosa.load(audiofile, sr = None)
    duration = librosa.get_duration(path=audiofile, sr=sr)
    return test_audio, sr, duration

#-----FILTERS-----
-----

def band_pass_filter(signal, sr, low_cutoff, high_cutoff):
    b, a = butter(N=3, Wn = [low_cutoff, high_cutoff], btype='band',
fs=sr)
    return lfilter(b, a, signal)

#-----FUNCTIONS FOR AUDIO PROCESSING-----
-----

```

```

def spectral_flux(signal):

    # Calcular el espectrograma de magnitudes
    spectrogram = np.abs(librosa.stft(signal, n_fft = FRAME_SIZE,
hop_length = HOP_SIZE))

    # Calcular el flujo espectral
    spectral_flux_values = np.sum(np.diff(spectrogram, axis=1)**2,
axis=0)

    return spectral_flux_values
def process(audio_in, audio_out, rms_umbral = 0.043, flux_umbral =
0.096):
    signal, sr, _ = load_audio(audio_in)

    rms = librosa.feature.rms(y = signal, frame_length = FRAME_SIZE,
hop_length = HOP_SIZE)
    rms /= np.max(np.abs(rms))
    trms = librosa.times_like(rms, sr = sr, hop_length = HOP_SIZE, n_fft
= FRAME_SIZE)
    trms /= trms[-1]

    flux = spectral_flux(signal)
    flux /= np.max(np.abs(flux))
    fluxframes = range(len(flux))
    tflux = librosa.frames_to_time(fluxframes, hop_length=HOP_SIZE, n_fft
= FRAME_SIZE)
    tflux /= tflux[-1]

    left_index = np.argmax(np.abs(flux) > flux_umbral)
    right_index = len(flux) - 1 - np.argmax(np.abs(np.flip(flux)) >
flux_umbral)

    tsignal = librosa.times_like(signal, sr = sr, hop_length=HOP_SIZE,
n_fft=FRAME_SIZE)
    tsignal /= tsignal[-1]

    flag      = False
    pad_left  = 0
    pad_right = 0
    flag_left = False
    flag_right = False

    while not flag:
        if rms[0, left_index] > rms_umbral:
            if left_index > pad_left + 15:
                rms_left = left_index - np.argmax(np.flip(np.abs(rms[0,
:left_index])) < rms_umbral)
            if rms_left <= 0:
                rms_left = left_index
                flag_left = True
        else:
            pad_left += 15

```

```

        left_index = pad_left + np.argmax(np.abs(flux[pad_left:]))
> flux_umbral)
    else:
        rms_left = left_index
        flag_left = True

        if rms[0, rigth_index] > rms_umbral:
            if rigth_index < (len(flux) - 1 - pad_rigth-15):
                rms_rigth = rigth_index + np.argmax(np.abs(rms[0,
rigth_index:])) < rms_umbral)
                if rms_rigth >= len(flux):
                    rms_rigth = rigth_index
                    flag_rigth = True
            else:
                pad_rigth += 15
                rigth_index = len(flux[:-pad_rigth]) - 1 -
np.argmax(np.flip(np.abs(flux[:-pad_rigth])) > flux_umbral))
        else:
            rms_rigth = rigth_index
            flag_rigth = True

        flag = flag_left and flag_rigth

        left_index = min(left_index, rms_left)
        rigth_index = max(rigth_index, rms_rigth)
        mask = tsignal >= tflux[left_index]
        ttrimmed = tsignal[mask]
        trimed = signal[mask]
        mask = ttrimmed <= tflux[rigth_index]
        ttrimmed = ttrimmed[mask]
        trimed = trimed[mask]

        sf.write(audio_out, trimed, sr)
#-----FUNCTIONS TO EXTRACT FEATURES-----
-----

def calculate_split_frequency_bin(split_frequency, sample_rate,
num_frequency_bins):
    """Infer the frequency bin associated to a given split frequency."""

    frequency_range = sample_rate / 2
    frequency_delta_per_bin = frequency_range / num_frequency_bins
    split_frequency_bin = math.floor(split_frequency /
frequency_delta_per_bin)
    return int(split_frequency_bin)

def band_energy_ratio(spectrogram, split_frequency, sample_rate):
    """Calculate band energy ratio with a given split frequency."""

    split_frequency_bin = calculate_split_frequency_bin(split_frequency,
sample_rate, len(spectrogram[0]))
    band_energy_ratio = []

    # calculate power spectrogram
    power_spectrogram = np.abs(spectrogram) ** 2

```

```

power_spectrogram = power_spectrogram.T

# calculate BER value for each frame
for frame in power_spectrogram:
    sum_power_low_frequencies = frame[:split_frequency_bin].sum()
    sum_power_high_frequencies = frame[split_frequency_bin:][].sum()
    band_energy_ratio_current_frame = sum_power_low_frequencies / (sum_power_high_frequencies + sum_power_low_frequencies)
    band_energy_ratio.append(band_energy_ratio_current_frame)

return np.array(band_energy_ratio)
def rms(signal, frames, hop):
    return librosa.feature.rms(y=signal, frame_length = frames,
hop_length = hop)
-----FEATURES EXTRACTION FUNCTION-----
-----
def get_features(signal, sr):
    feature = np.empty((1, 0))

    # BER
    spec = librosa.stft(signal, n_fft = FRAME_SIZE, hop_length =
HOP_SIZE)
    # max
    split_frequency = 600
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.max(np.abs(BER))
    feature = np.append(feature, feat)
    # min
    # 1
    split_frequency = 1900
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.min(np.abs(BER))
    feature = np.append(feature, feat)
    # 2
    split_frequency = 5000
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.min(np.abs(BER))
    feature = np.append(feature, feat)
    # 3
    split_frequency = 9000
    BER = band_energy_ratio(spec, split_frequency, sr)
    feat = np.min(np.abs(BER))
    feature = np.append(feature, feat)
    # std
    # 1
    split_frequency = 8000
    BER = band_energy_ratio(spec, split_frequency, sr)
    BER /= np.max(np.abs(BER))
    feat = np.std(BER) / np.mean(np.abs(BER))
    feature = np.append(feature, feat)
    # 2
    split_frequency = 1000
    BER = band_energy_ratio(spec, split_frequency, sr)

```

```

BER /= np.max(np.abs(BER))
feat = np.std(BER)/np.mean(np.abs(BER))
feature = np.append(feature, feat)

#ZCR
cutoff = 5000
cuton = 1000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered,
frame_length=FRAME_SIZE, hop_length=HOP_SIZE)[0]
zcr /= np.max(np.abs(zcr))
# mean
feat = np.mean(zcr)
feature = np.append(feature, feat)
# maximum
cutoff = 10000
cuton = 10
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered,
frame_length=FRAME_SIZE, hop_length=HOP_SIZE)[0]
feat = np.max(np.abs(zcr))
feature = np.append(feature, feat)
# std
cutoff = 10000
cuton = 20
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered,
frame_length=FRAME_SIZE, hop_length=HOP_SIZE)[0]
feat = np.std(zcr)/np.mean(np.abs(zcr))
feature = np.append(feature, feat)
# mean local
cutoff = 5000
cuton = 1000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered,
frame_length=FRAME_SIZE, hop_length=HOP_SIZE)[0]
zcr /= np.max(np.abs(zcr))
feat = np.mean(zcr[((len(zcr)*3)//14 - 5) : ((len(zcr)*3)//14 + 5)])
feature = np.append(feature, feat)
# local max
cutoff = 10000
cuton = 10
filtered = band_pass_filter(signal, sr, cuton, cutoff)
zcr = librosa.feature.zero_crossing_rate(filtered,
frame_length=FRAME_SIZE, hop_length=HOP_SIZE)[0]
feat = np.max(zcr[((len(zcr)*3)//4 - 10) : ((len(zcr)*3)//4 + 10)])
feature = np.append(feature, feat)

# Roll off
cuton = 100
cutoff = 8500
filtered = band_pass_filter(signal, sr, cuton, cutoff)

```

```

    roll_off = librosa.feature.spectral_rolloff(y=filtered, sr=sr,
n_fft=FRAME_SIZE, hop_length=HOP_SIZE, roll_percent=0.28)[0]
    roll_off /= np.max(np.abs(roll_off))
    # mean
    feat = np.mean(np.abs(roll_off))
    feature = np.append(feature, feat)
    # max
    cuton = 100
    cutoff = 8500
    filtered = band_pass_filter(signal, sr, cuton, cutoff)
    roll_off = librosa.feature.spectral_rolloff(y=filtered, sr=sr,
n_fft=FRAME_SIZE, hop_length=HOP_SIZE, roll_percent=0.55)[0]
    feat = np.max(np.abs(roll_off))
    feature = np.append(feature, feat)
    # std
    cutoff = 8500
    cuton = 50
    filtered = band_pass_filter(signal, sr, cuton, cutoff)
    roll_off = librosa.feature.spectral_rolloff(y=filtered, sr=sr,
n_fft=FRAME_SIZE, hop_length=HOP_SIZE, roll_percent=0.28)[0]
    roll_off /= np.max(np.abs(roll_off))
    feat = np.std(np.abs(roll_off))/np.mean(np.abs(roll_off))
    feature = np.append(feature, feat)

#MFCCS
n_mfcc = 4
# 1
cuton = 500
cutoff = 5000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
mfccs = librosa.feature.mfcc(y = signal, sr=sr, n_mfcc = n_mfcc,
n_fft = FRAME_SIZE, hop_length = HOP_SIZE)
feat = np.max(mfccs, axis = 1)
feature = np.append(feature, feat[3])

# 2
cuton = 10
cutoff = 8000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
mfccs = librosa.feature.mfcc(y = filtered, sr=sr, n_mfcc = n_mfcc,
n_fft = FRAME_SIZE, hop_length = HOP_SIZE)
mfccs /= np.max(np.abs(mfccs), axis = 1, keepdims=True)
feat = np.std(np.abs(mfccs), axis = 1)/np.mean(np.abs(mfccs), axis =
1)
feature = np.append(feature, feat[3])

# 3
cuton = 10
cutoff = 8000
filtered = band_pass_filter(signal, sr, cuton, cutoff)
mfccs = librosa.feature.mfcc(y = filtered, sr=sr, n_mfcc = n_mfcc,
n_fft = FRAME_SIZE, hop_length = HOP_SIZE)
mfccs /= np.max(np.abs(mfccs), axis = 1, keepdims=True)

```

```

mfccs = mfccs[:, ((mfccs.shape[1]*4) // 5 - 10):(mfccs.shape[1]*4)
//5 + 10)]
feat = np.std(np.abs(mfccs), axis=1) / np.mean(np.abs(mfccs), axis=1)
feature = np.append(feature, feat[1])

#envelope
env = rms(signal, FRAME_SIZE, HOP_SIZE)
env = env.reshape(-1,)
selected = np.linspace(0, len(env) - 1, 30, dtype=int)
env = env[selected]
feat = env[11]
feature = np.append(feature, feat)
feat = env[12]
feature = np.append(feature, feat)

return feature

#-----AUDIO RECORDING-----
-----
duration = 2.5      # Duración de la grabación en segundos
fs       = 48000     # Frecuencia de muestreo en Hz
#-----LOOP-----
-----
while True:
    c = input('Presione una ENTER cuando este listo para grabar o
presione \'n\' para salir.')
    if c.lower() == 'n':
        break
    os.system('cls')
    print("Grabando...")
    data_recorded = sd.rec(int(duration * fs), samplerate = fs, channels
= 1, dtype = 'int16')
    sd.wait()
    print("Grabación completa.")

    orden = os.path.join(original_path,
f"test{len(os.listdir(original_path)) + 1}.wav")
    sf.write(orden, data_recorded, fs)

#-----PROCESSING-----
-----
processed_order = os.path.join(processed_path,
f"test{len(os.listdir(processed_path)) + 1}.wav")
process(orden, processed_order)
#-----FEATURES EXTRACTION-----
-----
signal, sr, _ = load_audio(processed_order)
feature      = get_features(signal, sr)
test_features = feature.reshape(1, -1)
#-----LOAD THE TRAINED MODEL-----
-----
model          = joblib.load(model_file)

```

```

reduced:dict = model['features']
pca         = model['pca']
scaler      = model['scaler']
#-----COORDINATES TRANSFORMATION-----
-----
scaled_test_features = scaler.transform(test_features)
reduced_test         = pca.transform(scaled_test_features)
#-----PREDICTION-----
-----
prediction = knn(reduced, reduced_test, 3)
place = [shelf for shelf, fruit in shelfs.items() if fruit in prediction]
#-----RESULTS-----
-----
if len(place) == 0:
    print(f"No se encontró una {prediction} en ninguna de las 4 estanterías.")
elif len(place) == 1:
    print(f"Se encontró una {prediction} en el {place[0]}")
elif len(place) > 1:
    print(f"Se encontró una {prediction} en las siguientes estanterías: {place}")

fig, axs = plt.subplots(1, 4, figsize = (15, 5))
i = 0
for shelf, file in shelf_files.items():
    image = cv2.imread(file)
    axs[i].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    axs[i].axis('off')
    axs[i].set_title(shelf)

    if shelf in place:
        h, w, _ = image.shape
        rect = plt.Rectangle((0, 0), w, h, linewidth = 5, edgecolor = 'green', facecolor = 'none')
        axs[i].add_patch(rect)
    i += 1

plt.subplots_adjust(wspace = 0.5)
plt.show()
warnings.resetwarnings()

```