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 extraidas
  - La lista con la asignación de cada archivo a un cluster

LIBRERIAS

In [ ]:
```
import os
import numpy as np
import cv2
import matplotlib.pylab 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(o
processed  = [os.path.join(processed_path, image) for image in os.listdir(
```

KMEANS

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

    # kmeans++.
    # Seleccion 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 centroi

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

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

```
        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 recalculo 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 condicion 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 cluste
    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], r

    # 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],
```

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 rep
```

REPRESENTACIÓN DE LOS CLUSTERS

Agrupamos las imágenes en los clusters

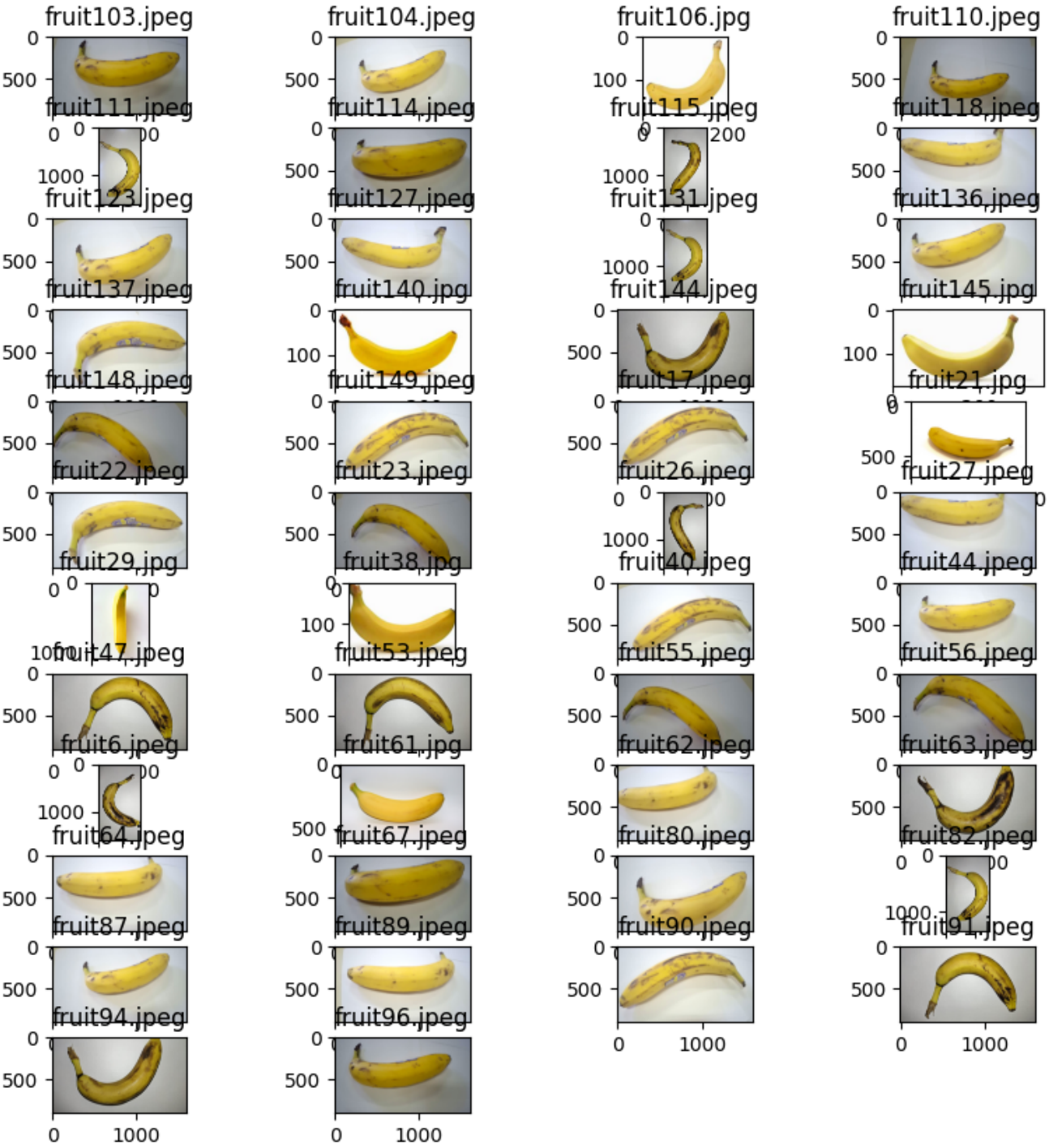
In [ ]:
```
labels_dict = dict(zip(names, [int(label) for label in labels]))
image_clusters = dict.fromkeys(set(labels))
for file, label in labels_dict.items():
    if image_clusters[label] is None:
        image_clusters[label] = []
    image_clusters[label].append(file)
```

Representación de las imágenes

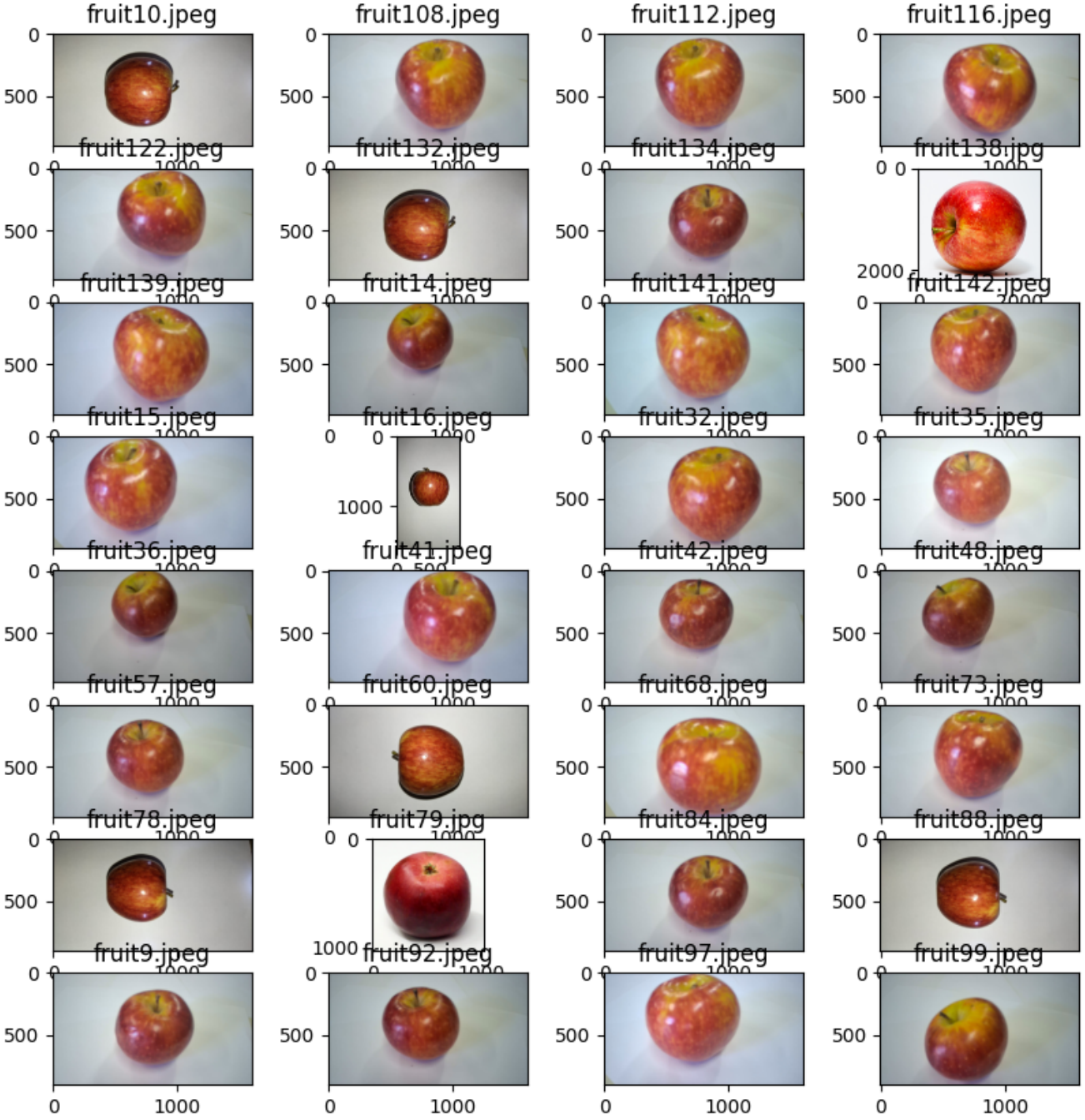
In [ ]:
```
for key, cluster in image_clusters.items():
    print(key)
```

```
    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()
```

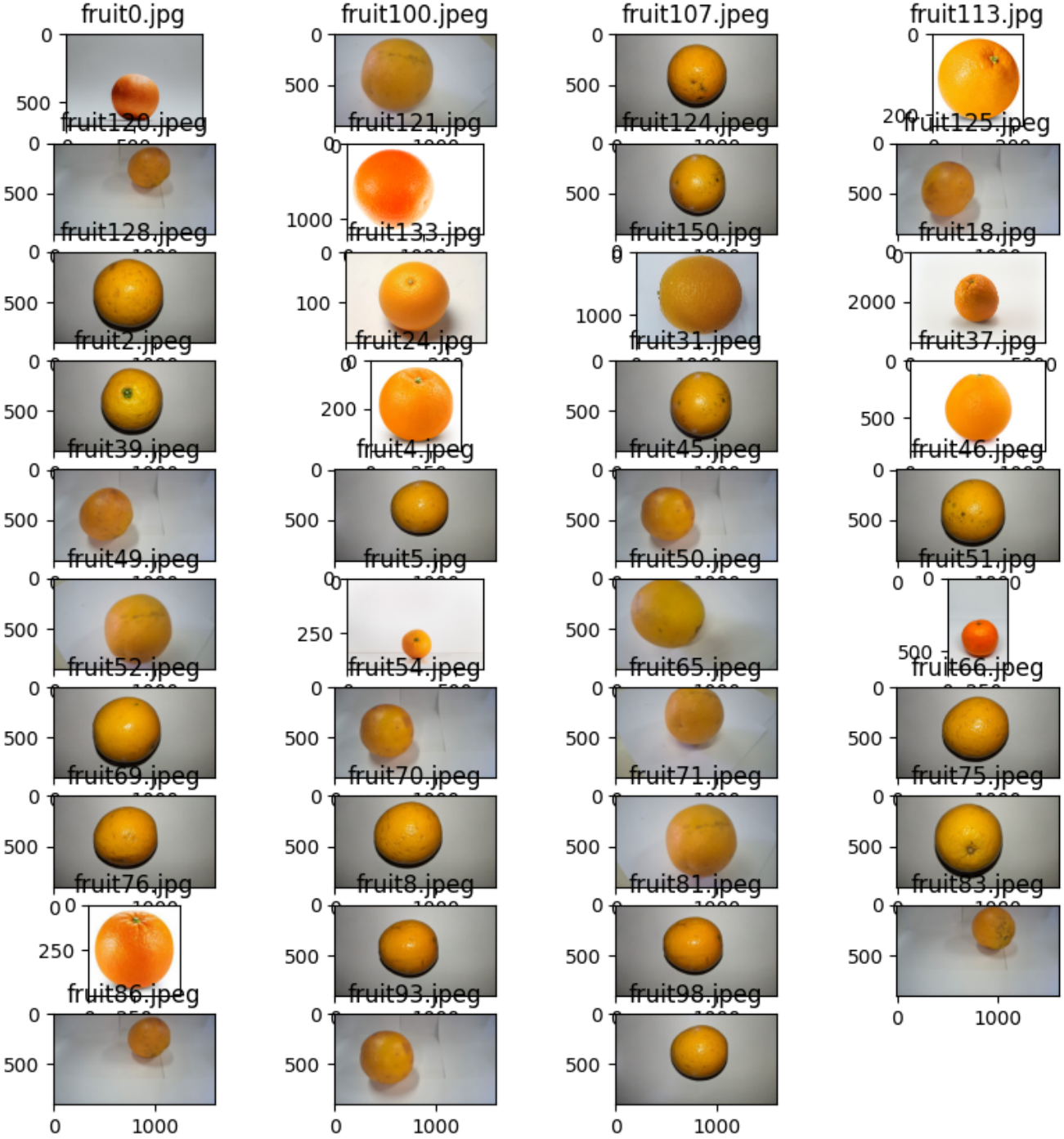
0.0



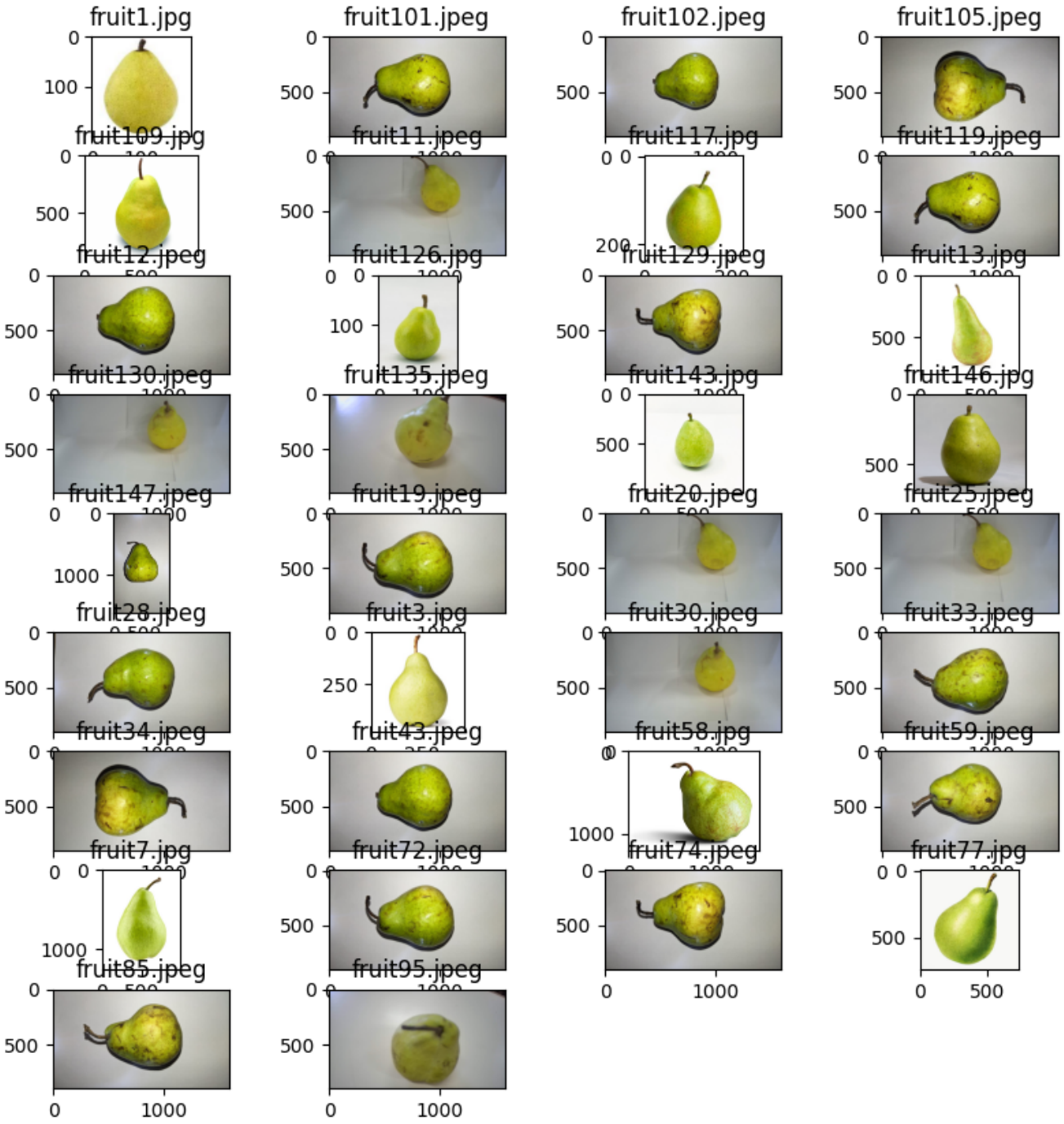
1.0


fruit10.jpeg
fruit108.jpeg
fruit112.jpeg
fruit116.jpeg
fruit122.jpeg
fruit132.jpeg
fruit134.jpeg
fruit138.jpg
fruit139.jpeg
fruit14.jpeg
fruit141.jpeg
fruit142.jpeg
fruit15.jpeg
fruit16.jpeg
fruit32.jpeg
fruit35.jpeg
fruit36.jpeg
fruit41.jpeg
fruit42.jpeg
fruit48.jpeg
fruit57.jpeg
fruit60.jpeg
fruit68.jpeg
fruit73.jpeg
fruit78.jpeg
fruit79.jpg
fruit84.jpeg
fruit88.jpeg
fruit9.jpeg
fruit92.jpeg
fruit97.jpeg
fruit99.jpeg

```
2.0
```


fruit0.jpg
fruit100.jpeg
fruit107.jpeg
fruit113.jpg
fruit120.jpeg
fruit121.jpg
fruit124.jpeg
fruit125.jpeg
fruit128.jpeg
fruit133.jpg
fruit150.jpg
fruit18.jpg
fruit2.jpeg
fruit24.jpg
fruit31.jpeg
fruit37.jpg
fruit39.jpeg
fruit4.jpeg
fruit45.jpeg
fruit46.jpeg
fruit49.jpeg
fruit5.jpg
fruit50.jpeg
fruit51.jpg
fruit52.jpeg
fruit54.jpeg
fruit65.jpeg
fruit66.jpeg
fruit69.jpeg
fruit70.jpeg
fruit71.jpeg
fruit75.jpeg
fruit76.jpg
fruit8.jpeg
fruit81.jpeg
fruit83.jpeg
fruit86.jpeg
fruit93.jpeg
fruit98.jpeg

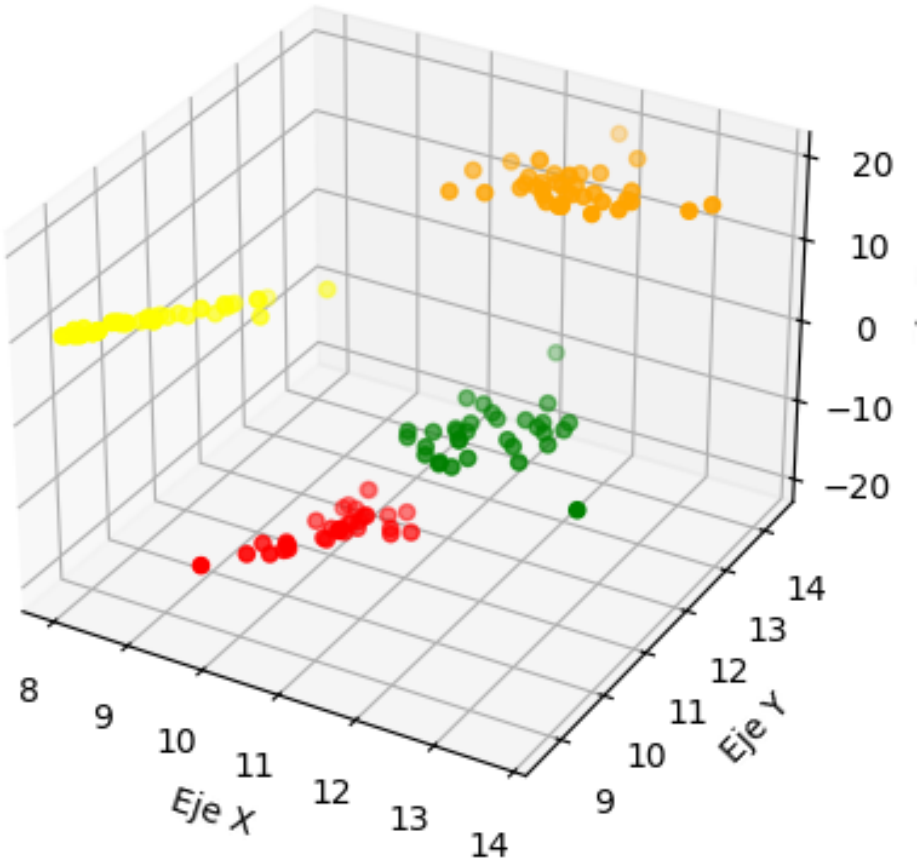
```
3.0
```



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, 'centroi
joblib.dump(data, training_data)
```

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