PREPROCESAMIENTO DE LAS IMAGENES

- Obtención de máscaras.
- Obtención de contornos que separan fruta de fondo.
- Las máscaras se guardan en archivos de imágen 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(
```

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 imágen

In [ ]:

```
for file in original:
    # BGR image
    image = cv2.imread(file)

    # Dimenssions
    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 pos 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 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
    kernel_dilatacion = np.ones((dilatacion_size,dilatacion_size), np.uint
    kmeans_mask       = cv2.dilate(eroded, kernel_dilatacion, iterations

    # Encontrar contornos
    kmeans_cnt, _ = cv2.findContours(kmeans_mask, cv2.RETR_EXTERNAL, cv2.C
    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)), tkme
```