Trabajo Final de Inteligencia Artificial I

Sistema de reconocimiento de frutas

Alumno: Cereda, Mariano

Legajo: 12254

Prof.: Dra. Ing. Selva S. Rivera

Introducción

Se busca diseñar un algoritmo de reconocimiento de frutas mediante el uso de técnicas de inteligencia artificial de aprendizaje supervisado (Knn) y no supervisado (Kmeans). Para el desarrollo de los mismos se utiliza el lenguaje de programación Python, ya que este contiene librerías que son de utilidad para el procesamiento de imágenes, como "OpenCV".

En cuanto al algoritmo en sí, previo al desarrollo de la lógica de IA correspondiente, se realiza un procesamiento en las imágenes: se aplican filtros correspondientes para poder extraer de forma correcta las características de cada una. Luego, los algoritmos desarrollados para el reconocimiento son "Knn" y "Kmeans".

El sistema busca reconocer 4 tipos diferentes de frutas: Bananas, limones, naranjas y tomates.

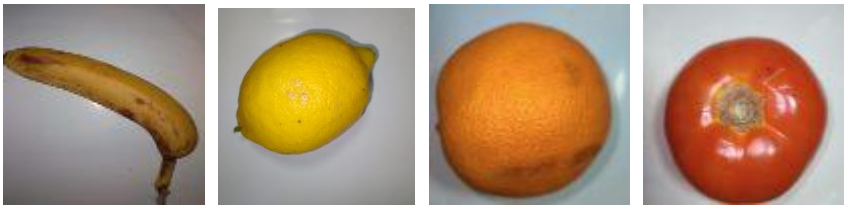

Figura (1)

Se utilizaron imágenes de internet y de elaboración propia para realizar el reconocimiento. La selección de estas es fundamental, ya que dependiendo de la calidad de las mismas varía mucho el resultado final.

Cabe aclarar que si se cuenta con una gran cantidad de datos de entrada (no es el caso de este trabajo), se puede optar por un sistema de reconocimiento que haga uso de redes neuronales convolucionales, siendo este un método de "Deep learning".

Figura (2)

En este trabajo se tiene en cuenta la curva roja mostrada en la figura (2), ya que los algoritmos de “Knn” y “Kmeans” presentan una buena “performance” cuando la cantidad de datos disponibles no es muy elevada.

Especificaciones del Agente

Un agente es una entidad capaz de percibir información de su entorno (a través de sensores), procesar la misma y actuar de forma que se maximice un resultado esperado. Por eso es importante definir el REAS del agente: Rendimiento, Entorno, Actuadores y Sensores.

REAS

Agente	Medida de rendimiento	Entorno	Actuadores	Sensores
Sistema de reconocimiento de frutas por visión artificial	Maximizar la precisión de reconocimiento. Constancia de resultados. Generalización. Minimizar errores de reconocimiento.	Caja de una verdulería o supermercado.	Caja registradora que cobra según lo que reconoce y cinta transportadora.	Cámara para tomar imágenes de las frutas.

Propiedades del entorno de trabajo

Entorno	Observable	Determinista	Episódico	Discreto	Agentes	Estático
Contene-dor de fruta	Parcialmente	Estocástico	Si	Si	Individual	Si

Diseño del agente

Elección de criterios de clasificación

Previo a comenzar el desarrollo del algoritmo, se debió decidir cuáles iban a ser los criterios que se iban a tener en cuenta para el reconocimiento de las diferentes frutas. Investigando y probando, se pudo concluir que 2 características significativas pueden ser:

- Momentos HU: Son momentos obtenidos a partir de los momentos normales de una imagen. Ambos (momentos HU y normales) son momentos de imagen, que se definen como un promedio ponderado particular de las intensidades de los píxeles.

- Componentes RGB: El color del píxel de una imagen se puede ver como la combinación de los colores rojo (R-“red”), verde (G-“Green”) y azul (B-“blue”). Si se toma la media de las componentes RGB de las imágenes de las frutas, se puede observar que es una característica que ayuda a discriminar bastante entre las diferentes frutas.

Investigación de librerías para procesamiento de imágenes.

Posteriormente, se investigó para determinar cuál iba a ser el procesamiento necesario en cada una de las imágenes para así obtener las 2 características previamente nombradas. Para ello se eligió la librería “OpenCV” de Python, la cual posee funciones adecuadas para esta tarea.

Además, para el cálculo de los momentos HU se decidió utilizar la librería “sickit-image” la cual posee métodos específicos para el cálculo de momentos de imagen.

Obtención de imágenes adecuadas

Si bien existen grandes bases de datos en sitios web como “kaggle” para incentivar proyectos de IA y brindar información, se optó por tomar imágenes propias para el entrenamiento del algoritmo. Para ello, se buscó iluminar de la mejor forma posible el entorno de la imagen y así poder minimizar posibles sombras que interfieran en el reconocimiento. Además, se tuvo en cuenta que la imagen generada sea cuadrada para luego poder realizar un redimensionamiento de la misma y que su tamaño sea de 100x100 pixeles. Este tamaño de imagen agiliza el procesamiento, ya que el peso de las imágenes es pequeño (aproximadamente 3KB).

Normalizado de los datos

Para poder llevar a cabo una comparación correcta entre los datos se los normaliza teniendo en cuenta los valores máximos y mínimos (tanto de media RGB como de momentos HU). Si esto no se realiza se pueden tener datos de RGB o momentos HU fuera de escala, pudiendo realizar una comparación errónea y haciendo que algunos datos tengan más relevancia que otros numéricamente.

Desarrollo de algoritmo de “Knn”

Teniendo toda la información necesaria para el reconocimiento, se procedió a programar uno de los algoritmos de IA: Knn (“k-nearest-neighbours” por sus siglas en inglés). Como su nombre indica, el algoritmo considera los “K” (siendo este un número natural) vecinos más cercanos (se entiende por “cercano” a los puntos de “training” con características más similares) al dato que se desea clasificar.

Es decir, se obtiene la media de RGB de la imagen a clasificar y la suma de los momentos HU 1 y 3, obteniendo así un "punto" dato (el cual se puede graficar ya que se trabaja con 2 variables). Luego, se calcula la distancia entre dicho punto y todos los demás obtenidos durante el "train" del algoritmo. Finalmente, se consideran los "k" puntos para los cuales se obtuvo la menor distancia y, dependiendo de la clase a la cual pertenecen estos vecinos más cercanos, es la clase que se le asignará a la imagen de test.

Desarrollo de algoritmo de "Kmeans"

Es un algoritmo de clasificación no supervisada que agrupa objetos en "k" grupos basándose en sus características, donde estas son los momentos HU de las imágenes y las medias de colores RGB. Al principio se deben inicializar en algún determinado valor los centroides de cada uno de los k grupos. Luego, a cada punto dato se le asigna el centro más cercano. Finalmente, se actualiza la posición de los centroides de cada grupo tomando como nuevo centro la posición del promedio de los objetos pertenecientes a dicho grupo.

Corrección de hiperparámetros

Una vez realizados los algoritmos de reconocimiento en sí, se corrigieron algunos parámetros como la constante "k" del algoritmo "Knn" y los valores de inicialización de centroides en el de "Kmeans". Esto se hace según los resultados obtenidos para cada uno de estos hiperparámetros.

Especificaciones de Código

El algoritmo en Python se divide en 5 módulos o archivos:

- **Módulo "VisionArtificial_main.py":** Es el encargado de ejecutar el bucle principal, es decir que su tarea es ir instanciando los objetos que correspondan e ir ejecutando las funciones contenidas en los demás módulos. Se puede dividir en 4 bucles: el primero de procesamiento de imágenes de "train", el segundo de procesamiento de imágenes de test, el tercero del algoritmo de "Knn" y el último del algoritmo de "Kmeans".

- **Módulo "PreprosImg.py":** Dicho módulo contiene una clase denominada "Preprocesamiento_img", la cual posee 2 métodos principales: El denominado "ChargeImage" se encarga de cargar en una lista las imágenes de training y test, y el otro denominado "Convert" que realiza el "resize" de las imágenes, además de ejecutar 2 filtros sobre la imagen original: suavizado de texturas y conservación de bordes bien nítidos (cv2.edgePreservingFilte), y la obtención de la imagen en escala de grises.

```python
def Convert(self,src,n):
    'Hace un pequeño preprocesamiento para luego procesar directamente estas imagenes'

    ###############LECTURA DE LA IMAGEN ORIGINAL#######################
    imOriginal = cv2.imread(src,1)
    ratio= 100/imOriginal.shape[1]                                    # new ratio
    dim = (100,int(imOriginal.shape[0]*ratio))                         # new dimensio
    imOriginal = cv2.resize(imOriginal,dim,interpolation=cv2.INTER_AREA)

    ################ESTILIZAMOS LA IMÁGEN ############################
    imSty = cv2.stylization(imOriginal, sigma_s=30, sigma_r=0.8)
    src_sty = "Imagenes/Estilizadas/Img_"+str(n)+"_sty.jpg"
    cv2.imwrite(src_sty,imSty)
    imEp = cv2.edgePreservingFilter(imSty, flags=1, sigma_s=30, sigma_r=0.8)
    src_ep = "Imagenes/Estilizadas/Img_"+str(n)+"_ep.jpg"
    cv2.imwrite(src_ep,imEp)

    ################CONVERTIMOS A ESCALA DE GRISES###################
    imGris = cv2.imread(src,0)

    return imGris,imOriginal,imEp
```

Figura (3)

A continuación se muestran 2 ejemplos de imágenes con el filtro de suavizado (denominada imEp en el código):

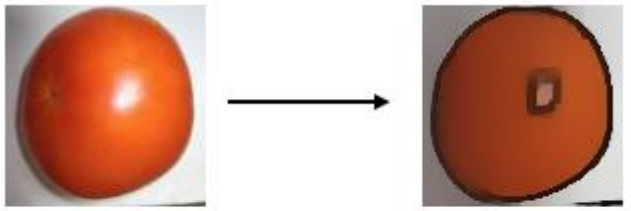

Imagen original *Imagen suavizada*

Figura (4)

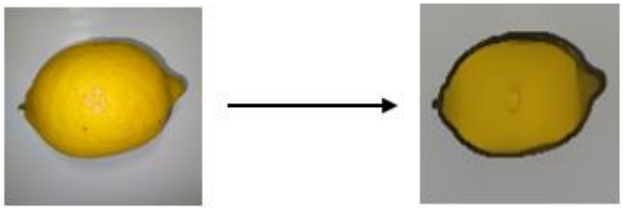

Imagen original *Imagen suavizada*

Figura (5)

- **Modulo "ProcImg.py":** Contiene una clase denominada "Procesamiento_img" cuyos métodos realizan varias tareas: aplican más filtros a las imágenes, realizan el cálculo de los momentos HU, separan la imagen de la fruta de su fondo y obtiene los valores de los canales RGB y además, de ser necesario, podría obtener otras propiedades como perímetros, área y redondez (aunque finalmente no se utilizaron). En la siguiente figura se muestra el llamado a estos métodos desde el modulo principal:

```
while n<len(srcTotal):
    #PREPROCESAMIENTO de imágenes - devuelve la imágen en gris , la origina
    (imGrey,imOriginal,imEp)=preprocImg.Convert(srcTotal[n],n)

    # COMIENZA EL PROCESAMIENTO

    ##Aplicacion de filtro gaussian blur y binarización de la imagen
    imGauss = img.GaussianFilter(imGrey)
    imBinaria = img.BinarizarImagen(imGauss)

    ##Separación de la fruta y el fondo
    (imBinary,cnts,imMask,imFruit)=img.SeparateBackground(imGauss,imEp, n)

    #Normalizado de la imagen
    imNorm = img.NormalizeImage(imBinary)

    #Extracción de capas e histogramas
    red_image = img.GenerarCapas("rojo",imFruit)
    green_image = img.GenerarCapas('verde',imFruit)
    blue_image = img.GenerarCapas('azul',imFruit)
    (hist,meanRGB) = img.CalculateHistogram(imFruit,n)
    means_RGB_data.append(meanRGB)

    # Calculo de momentos hu
    (sum31,mhu3,mhu1,mhuActual)= img.HUmoments(imNorm)
    mhus_sum31_data.append(sum31)
```

Figura (6)

Aquí hay 2 imágenes importantes obtenidas durante el filtrado: La denominada *ImBinary*, que luego de ser normalizada es usada para el cálculo de los momentos HU. Por otro lado, la denominada *ImFruit* que se utiliza para el cálculo de las media RGB.

A continuación se muestran algunos resultados para *ImBinary:*

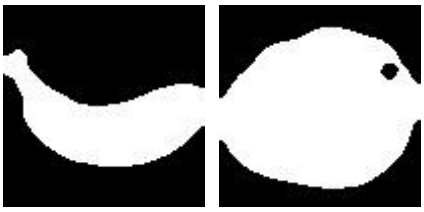

Figura (7) - *Imagén binarizada de banana y tomate*

Por otro lado, se muestran resultados para *ImFruit,* la cual contiene la imagen a color filtrada con un fondo negro aplicado*:*

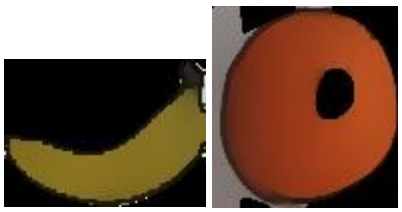

Figura (8) - Imagen de banana (izquierda) y naranja (derecha) a color y fondo negro

Ahora bien, el cálculo y suma de los momentos HU se realiza en el método "HUmoments()", que se muestra a continuación:

```
def HUmoments(self,imBinary):
    mu = skimage.measure.moments_central(imBinary)
    nu = skimage.measure.moments_normalized(mu)
    mhu =  skimage.measure.moments_hu(nu)
    for i in range(len(mhu)):
        mhu[i] = -1*np.sign(mhu[i])*np.log10(np.abs(mhu[i]))
    mhuActual = mhu.tolist()
    mhu1=mhu[0]
    mhu3=mhu[2]
    sum31 = mhu1 + mhu3
    #sumTot = sum(mhu)
    return sum31,mhu3,mhu1,mhuActual
```

Figura (9)

Se puede ver el uso de las funciones de cálculo de los distintos momentos de imagen de la librería "skimage.measure". Luego, recordando que para cada imagen se tienen 7 momentos, se elige el primero y tercero (más significativos y de mejor resultado) y se los suma.

Por otro lado, la obtención de las medias RGB se calcula dentro del método "CalculateHistogram ()":

```
def CalculateHistogram(self,imFruit,index):
    color = ["b","g","r"]
    hist = list()
    for i, c in enumerate(color):
        hist.append(cv2.calcHist([imFruit],[i], None, [255], [10, 250]))
    meanRGB = np.mean(hist)
    return hist,meanRGB
```

Figura (10)

Una vez realizado esto, se puede decir que ya se tienen las características de momentos HU y media RGB de cada foto.

- **Módulo "Knn.py"**: A partir de los datos obtenidos anteriormente, realiza el algoritmo de "knn". Se divide en 2 funciones: una de cálculo de las distancias euclidianas entre cada punto y el punto de test correspondiente, denominada "CalculateDistEucl ()". Otra denominada "CalculateClass ()" que se encarga de clasificar la fruta de test como banana, limón, naranja o tomate.

- **“Modulo “Kmeans.py”:**
  Es el encargado de ejecutar el algoritmo de reconocimiento que lleva su nombre. Posee 2 métodos: El denominado “calculate_centroids” que calcula los centroides de cada una de las clases. El otro llamado “calculate” que, dependiendo de los centroides calculados, asigna clases a las frutas de “test”.

Ejemplo de aplicación y análisis de resultados

Para un ejemplo de aplicación, se utilizaron las siguientes imágenes para “test”:

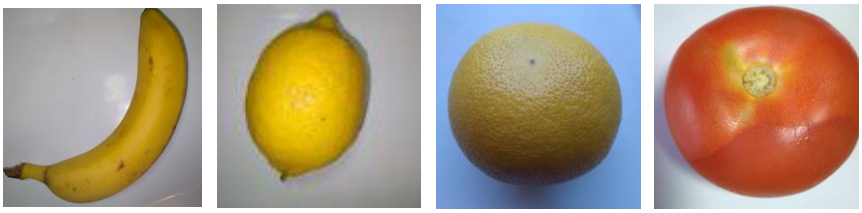

Figura (11) – Imágenes utilizadas para “test” en primer ejemplo.

Y la gráfica obtenida donde se observan las características de todas las imágenes (test y training) es la siguiente:

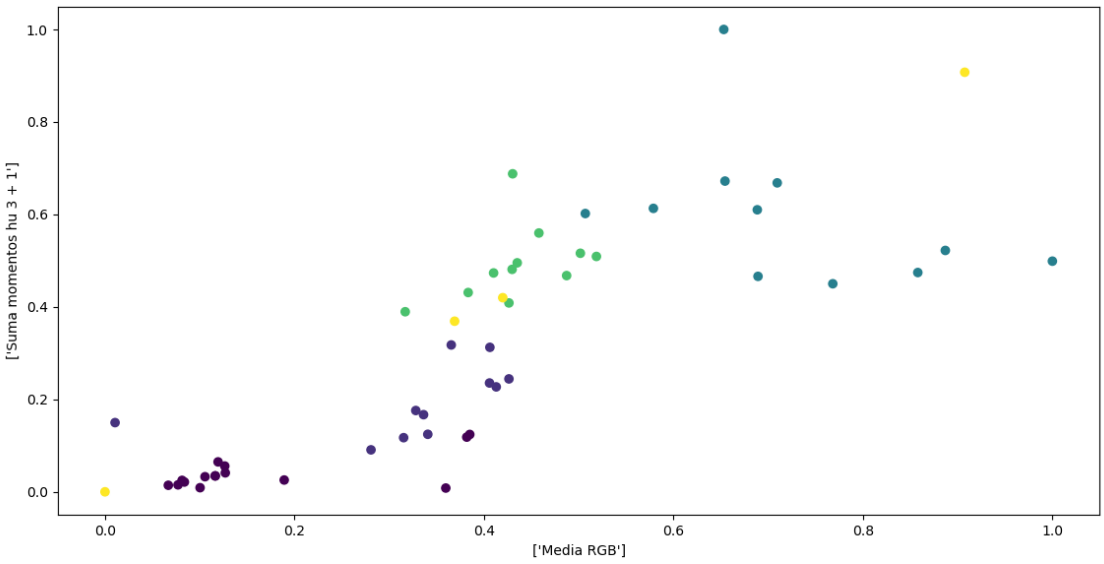

Figura (12) – Gráfica de puntos de “train” y “test”

Donde en color amarillo se pueden ver los puntos que representan a las imágenes de test.

Los resultados fueron los siguientes:

```
 RESULTADOS SEGÚN KNN:

Fruta real:  Imagenes/Test/banana.jpg
Fruta detectada:Banana
Fruta real:  Imagenes/Test/limon.jpg
Fruta detectada:Limon
Fruta real:  Imagenes/Test/naranja.jpg
Fruta detectada:Naranja
Fruta real:  Imagenes/Test/tomate.jpg
Fruta detectada:Tomate

 RESULTADOS SEGÚN KMEANS:

[0, 2, 3, 3]
Frute detectada: Banana
Frute detectada: Tomate
Frute detectada: Naranja
Frute detectada: Naranja
```

Figura (13)

Para el algoritmo de “Knn” se obtuvo una efectividad del 100% utilizando un k = 4, acertando en la predicción de todas las frutas. Mientras que en el algoritmo de “Kmeans” la efectividad fue del 50%, acertando en la primer predicción (Banana) y en la tercera (Naranja).

Para mejorar el rendimiento del algoritmo de “Kmeans” se modificaron los valores de los centroides iniciales, logrando aumentar ahora la efectividad al 100%.

```
 RESULTADOS SEGÚN KMEANS:

Fruta detectada: Banana
Fruta detectada: Limon
Fruta detectada: Naranja
Fruta detectada: Tomate
```

Figura (14)

Ahora bien, en el siguiente caso de aplicación se tomaron otras imágenes para hacer el “test”. Se optó por fotos tomadas desde internet (como se muestra en la figura 15) y también de elaboración propia (figura 11).

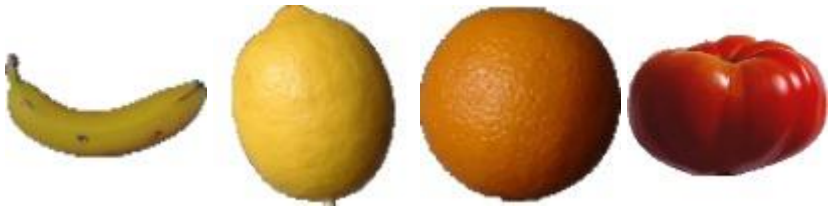

Figura (15) – Imágenes utilizadas para “test” en segundo ejemplo.

Se analizaron de esta manera 16 imágenes en total, obteniendo los siguientes resultados:

```
 RESULTADOS SEGÚN KNN:
Fruta real:  Imagenes/Test/banana_t1.jpg
Fruta detectada:Banana
Fruta real:  Imagenes/Test/banana_t2.jpg
Fruta detectada:Banana
Fruta real:  Imagenes/Test/banana_t3.jpeg
Fruta detectada:Banana
Fruta real:  Imagenes/Test/banana_t5.jpg
Fruta detectada:Banana
Fruta real:  Imagenes/Test/limon_t1.jpeg
Fruta detectada:Limon
Fruta real:  Imagenes/Test/limon_t2.jpg
Fruta detectada:Limon
Fruta real:  Imagenes/Test/limon_t3.jpg
Fruta detectada:Banana
Fruta real:  Imagenes/Test/limon_t5.jpeg
Fruta detectada:Limon
Fruta real:  Imagenes/Test/naranja_t1.jpg
Fruta detectada:Tomate
Fruta real:  Imagenes/Test/naranja_t2.jpg
Fruta detectada:Naranja
Fruta real:  Imagenes/Test/naranja_t3.jpeg
Fruta detectada:Naranja
Fruta real:  Imagenes/Test/naranja_t5.jpeg
Fruta detectada:Naranja
Fruta real:  Imagenes/Test/tomate_t1.jpg
Fruta detectada:Tomate
Fruta real:  Imagenes/Test/tomate_t2.jpg
Fruta detectada:Tomate
Fruta real:  Imagenes/Test/tomate_t5.jpeg
Fruta detectada:Tomate
```

Figura (16)

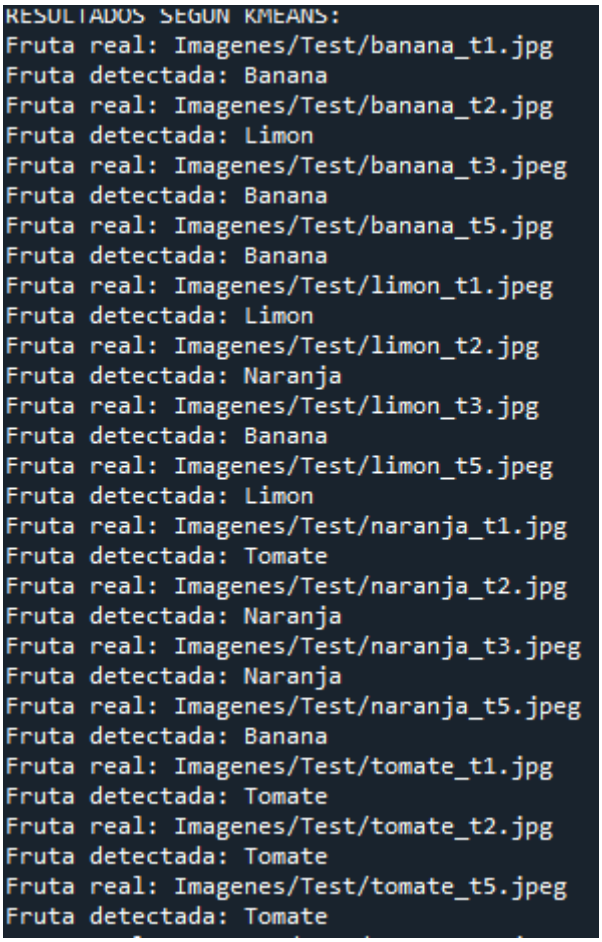

Figura (17)

Se puede observar que el algoritmo de “knn” obtiene una eficacia del 87.5%(14 clasificadas correctas sobre 16), mientras que el “kmeans” de 75% (12 clasificadas correctas sobre 16). Si se analizan las imágenes donde se comete un error, son las que provienen de internet y no son de elaboración propia. Por eso, es importante resaltar la importancia y relación que debe existir entre las imágenes de “train” y “test”.

Por último, en la figura (18) se muestra la gráfica obtenida para el caso de aplicación mostrado. Donde los colores de los puntos representan:

- Azul: Grupo de limones de “train”
- Amarillo: Grupo de bananas de “train”.
- Gris: Grupo de naranjas de “train”.
- Violeta: Grupo de tomates de “train”.
- Verde: Grupo de frutas de “test”.

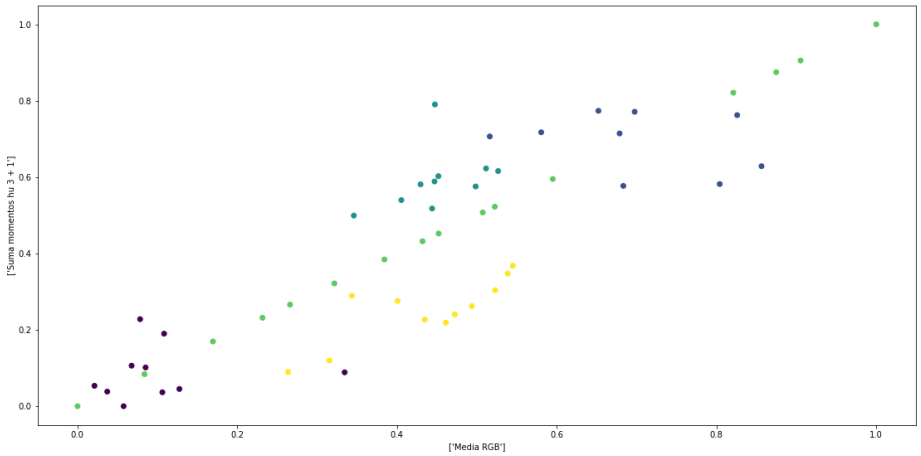

Figura (18) – Gráfica de puntos de “train” y “test” en segundo ejemplo.

Existe una separación bien marcada entre los grupos azul y amarillo, pero no ocurre lo mismo entre los grupos gris y violeta, donde la interfaz entre ambos es un tanto difusa y cercana, dando lugar a errores.

RESULTADOS

Lo más importante a destacar o apreciar es la varianza que se obtienen en los resultados dependiendo de la calidad de las imágenes, tanto de “train” como de “test”. Se observa que, al utilizar para el test imágenes también obtenidas con la cámara de fotos propia (se podría decir que son de la misma “calidad” que las utilizadas para “train”) los resultados son satisfactorios, mientras que cuando se utilizaron las imágenes de internet, los resultados empeoraron. Estos problemas se observaron en el reconocimiento de las naranjas principalmente, sin afectar a las demás frutas demasiado.

Por otro lado, se puede apreciar que en las zonas donde se “solapan” distintos grupos de frutas es debido a la calidad de la imagen, siendo estas afectadas por sombras o reflejos en algunas ocasiones.

Finalmente, también es importante destacar que una mayor cantidad de datos no mejora significativamente los resultados finales obtenidos. Esta característica es notable en otro tipo de algoritmos de “Deep learning” pero no en los utilizados en el presente trabajo.

CONCLUSIONES

Se pudo llegar a varias conclusiones luego de la realización del proyecto:

- La calidad de las imágenes utilizadas influye mucho en el resultado final del reconocimiento. Es importante tratar de utilizar imágenes lo más similares al caso de aplicación real. Si bien entonces, se podrían haber usado imágenes de mejor calidad u obtenidas de internet para todo el trabajo, esto no sería del todo consistente con la aplicación real de un supermercado. En esta situación, uno no puede editar o recortar las imágenes para un mejor resultado.
- Para los algoritmos utilizados, la cantidad de datos es adecuada. Colocar más imágenes para el "train" del algoritmo, diferente a lo que se puede pensar, puede llevar a resultados peores. Esto es debido a la dispersión de los datos. Por eso, lo recomendable es utilizar siempre imágenes parecidas a lo que puede ocurrir en una aplicación real.
- La modificación de hiperparámetros como la posición inicial de los centroides o el valor de "k" del algoritmo de knn es muy importante, ya que su correcta calibración puede llevar a grandes mejoras en el resultado del algoritmo.
- Para menos datos, "knn" se comporta mejor ya que "kmeans" depende del promedio de los datos.
- Es fundamental la normalización de la información obtenida.

Finalmente, en un caso real donde se busque una precisión más elevada y la cantidad de datos disponible puede ser mayor se podría recomendar el uso de una red neuronal convolucional. Estas han presentado muy buenos resultados en el reconocimiento de imágenes.