



UNIVERSIDADE  
DE VIGO



EET  
Campus Universitario de Vigo.  
36310 Vigo (Pontevedra).

---

**Fundamentos de Sonido e Imagen  
GETT (segundo año).**

## **Trabajo práctico II: Clasificación del color.**

**Resumen:** en esta práctica, construiremos un clasificador simple basado en información de color. Aunque simple, presentaremos algunos conceptos, claves para el tema "candente" de la visión por computadora, tales como: vector de características, métrica de distancia y matriz de confusión.

**Duración:** 2 sesiones.

### **1. Introducción:**

En esta práctica, veremos cómo con un simple conocimiento de Matlab y el procesamiento de imágenes, podemos desarrollar una aplicación de reconocimiento de imágenes interesante. El material se puede descargar desde [faiitc](http://faiitc).

### **2. El problema:**

La aplicación propuesta debe ser alimentada con imágenes como los ejemplos en la figura 1. Imagine que estamos trabajando en una planta procesadora de frutas, donde los artículos entrantes (frutas) son fotografiados por una cámara industrial mientras pasan por un cinta transportadora (en este caso cubierta con una tela negra).

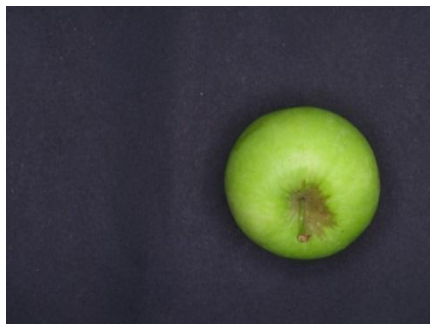


Figura 1. Imágenes de frutas.



Figura 2. Izquierda: Sistema industrial que estamos imitando. Derecha: montaje real para captura de imágenes.

Intentaremos resolver el siguiente problema "simple":

<< Desarrollar funciones matlab capaces de distinguir manzanas verdes y rojas >>

### 3. La solución (general):

Un esquema general para un problema de clasificación podría ser este:

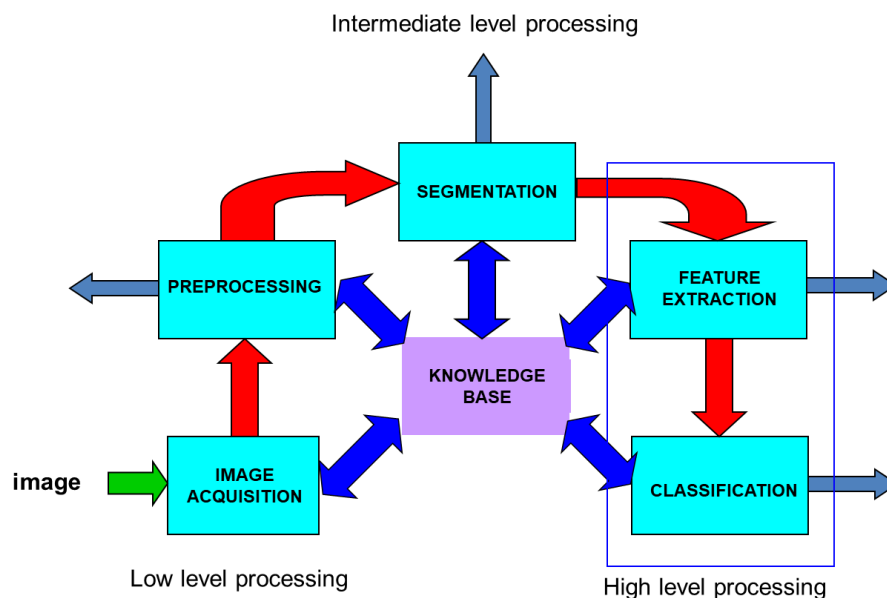


Figura 3. General esquema para aplicaciones de análisis de imágenes.

Donde las flechas rojas indican el flujo de información y las diferentes acciones pueden definirse más concretamente de la siguiente manera:

- Adquisición de imágenes: fácil de definir pero normalmente es la parte más importante. Una buena elección de cámara, buena iluminación, ausencia de sombras ... facilitará todos los procesos restantes.
- Preprocesamiento: acciones tomadas para mejorar las imágenes adquiridas. Puede consistir en borrado, mejora del color ... A veces los resultados no son una "mejora visual", pero lo que importa aquí es el contexto del problema.
- Segmentación: extracción de las regiones de interés (por ejemplo, la matrícula de un automóvil en una imagen de la calle).
- Extracción de características: consiste en obtener valores numéricos de una imagen (o de una región de interés de una imagen). Los valores como "nivel de gris medio", "variación de nivel de gris", "histograma de color" son ejemplos de características. Estas características son útiles si pueden usarse para la clasificación. Normalmente extraemos un conjunto de características que definen lo que se llama un "vector de características".

- Clasificación: significa procesar los vectores de características para obtener algún tipo de decisión. La mayoría de las veces consiste en determinar a qué clase pertenece la imagen original. En nuestro ejemplo, todas las imágenes pertenecen a una de dos clases (manzanas rojas o verdes); El problema puede ser mucho más complicado como, por ejemplo, un OCR (Reconocimiento Óptico de Caracteres) donde las imágenes deben segmentarse para obtener subimágenes que contengan diferentes caracteres y deben clasificarse en una de 26 clases diferentes (suponiendo un alfabeto de 26 letras).

Observe que cada uno de estos procesos diferentes puede producir resultados individuales que son útiles para algunas aplicaciones, por ejemplo: una imagen con color mejorado es un producto de interés en sí mismo.

#### 4. Método propuesto (particular):

Para este caso y con respecto al esquema general de la figura 3, proponemos el siguiente método (vea que lo describimos particularizando cada una de las etapas de procesamiento anteriores):

- Adquisición de imágenes: las imágenes se han capturado como en la Figura 2 (derecha), la cámara es una Canon Powershop SX710, la configuración de la cámara y la lente es la misma para todas las fotos, las condiciones de iluminación también son constantes. El artículo de fruta es único en cada foto. La posición no es la misma. Los artículos son diferentes. Quizás en algunas fotos, el elemento no está completo, pero al menos la mayor parte está presente.
- Preprocesamiento: basaremos el reconocimiento en color, por esta razón, aplicaremos aquí un método sencillo para mejorar el color.
- Segmentación: en este caso, la segmentación no será necesaria ya que utilizaremos un tipo de extracción de características que hace innecesario eliminar el fondo. Podríamos detectar el fondo, por ejemplo, convirtiendo imágenes en escala de grises y aplicando un umbral de luminancia.
- Extracción de características: en este caso, es la parte más importante. Utilizaremos histogramas de color porque: describen el color y **NO CAMBIAN** debido a las traslaciones y rotaciones de objetos. Además, el fondo negro causará un pico en la parte más baja de los histogramas, y centrarse en la parte correcta de los gráficos harán la segmentación innecesaria.
- Clasificación: tenemos 6 muestras por clase (6 manzanas verdes y 6 rojas). Elegiremos tres de cada clase como muestras de "**entrenamiento**" (utilizadas para enseñar al clasificador) y el resto serán muestras de "**prueba**". Usando las muestras de entrenamiento, calcularemos el vector de características para cada uno, y luego calcularemos un vector medio "por clase". Esos vectores medios se denominarán "**prototipos**". El conjunto de prototipos para todas las clases se llama "**alfabeto**". Al intentar clasificar una muestra de prueba, calcularemos su vector de características y lo llamaremos "**patrón**". El patrón de entrada debe compararse con todos los prototipos del alfabeto, el resultado más similar definirá la "**decisión de clasificación**". Para comparar vectores de características, definiremos una métrica o "**distancia**", la distancia mínima definirá la similitud máxima. Intentaremos con todas las muestras de prueba y resumiremos los resultados en una "**matriz de confusión**".

#### 5. Preprocesamiento (mejora del color):

En primer lugar, obtenga las imágenes de entrada en el archivo comprimido llamado **Fruits.rar**, disponible en faitic.

Ahora, desarrolle una función matlab de acuerdo con la siguiente definición:

**función imout = ColorEnhance2 (nombre de archivo)**

Donde el nombre de archivo de la variable de entrada debe ser un "nombre de archivo" de la imagen e imout debe ser una imagen RGB **doble**, con sus colores mejorados.

Para codificar la función, siga las siguientes instrucciones:

- Leer la imagen del archivo, esta operación obtiene imagen RGB **uint8**.
- Convierta la imagen en imagen RGB **doble**.
- Ponga la imagen en escala de grises correspondiente en una variable auxiliar.
- Obtenga valores de luminancia máximos y mínimos.

- Aplicar la ecuación:  $\mathbf{Im}_{out} = (\mathbf{Im}_{in} - \text{mínimo}) / (\text{máxima} - \text{mínima})$ .
- "Corregir valores fuera de rango". Id EST: si un valor de  $\mathbf{Im}_{out}$  es mayor que 1 debe forzarse a 1 valor, si un valor es menor que cero, forzarlo a cero.

Si se codifica correctamente, esta función se puede aplicar a cualquier imagen (fruta o no). Vea el siguiente ejemplo:



Figura 4. Ejemplo de mejora de color.

Sorprendentemente, el fondo parece "un poco azul". Esto probablemente se deba a un error de AWB (balance de blancos automático) en la cámara. Como no utilizaremos la parte inferior de los histogramas, esperamos que esto no nos ponga en problemas.

## 6. Extracción de características (cálculo de histograma de color):

Ahora, nos vamos a centrar en extraer características numéricas de una imagen. Como se indicó anteriormente, en este caso particular, estamos interesados en algún tipo de histograma de color. Como se define en el material teórico, se obtiene un histograma contando píxeles del "mismo" color (dividimos todo el intervalo en un número finito de intervalos para definir lo que se considerará como "el mismo color"). Nuestro primer problema es que la función de Matlab `imhist` solo calcula histogramas en escala de grises y nos enfrentamos a un problema de "color". Se pueden definir histogramas de color que consideren el espacio de color RGB como coordenadas tridimensionales. En este caso, los intervalos en escala de grises se convierten en regiones cúbicas. Usaremos algo mucho más simple. Tomando por separado los tres planos de color de una imagen RGB, podemos obtener tres histogramas en escala de grises: tres vectores.

Por ejemplo, para las dos imágenes en la figura 1, obtenemos los siguientes histogramas:

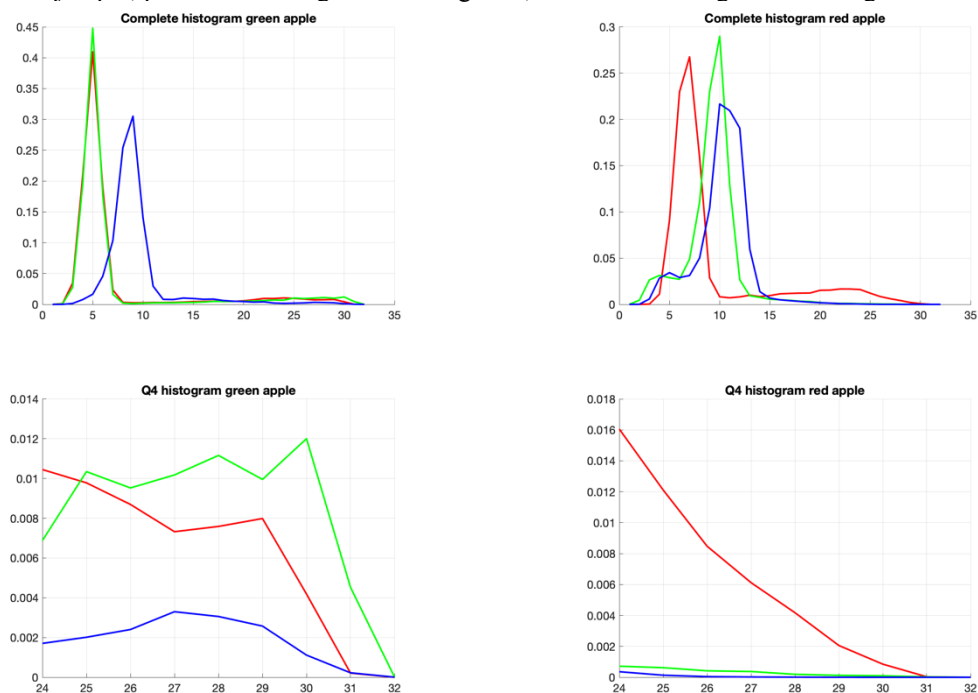


Figura 5. La fila de arriba muestra los histogramas completos, a continuación vemos los detalles del cuartil más a la derecha. Las curvas de una manzana verde están a la izquierda, derecha para una roja.

Las imágenes se han mejorado en color antes de calcular los histogramas. Hemos utilizado  $N = 32$  bins (intervalos) para cada uno. En la representación, hemos usado una curva roja para el componente rojo y así sucesivamente (de manera similar a los histogramas de color mostrados en Photoshop y aplicaciones similares). Estos histogramas están normalizados, esto significa: cada vector se divide por su propia suma, de modo que puede considerarse como un PDF aproximado (Función de Densidad de Probabilidad).

Con respecto a la figura 5, observe que los histogramas completos están dominados por los picos a valores bajos. Estos picos se deben a píxeles de fondo. En el caso de la manzana verde, la curva azul tiene el pico principal un poco desalineado debido al ligero color de fondo azul. Más importantes para nuestros propósitos son los detalles en los gráficos a continuación. Centrándonos en los últimos valores  $N / 4$  de cada histograma, encontramos lo que estábamos buscando. Vea cómo el componente verde domina en la manzana verde, mientras que el rojo gobierna para la manzana roja. El componente azul no es significativo en ninguno de los dos gráficos.

Con estos ingredientes en mente, diseñaremos una función de "extracción de características":

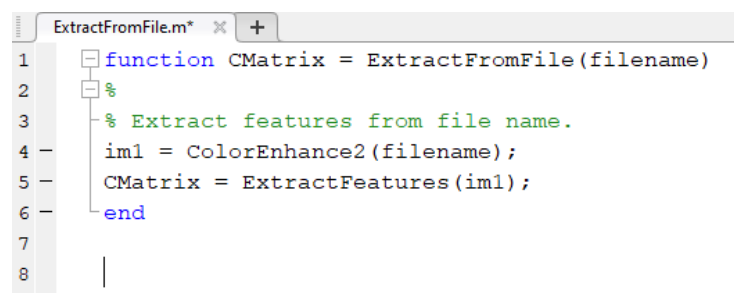
```
function feat = ExtractFeatures(im)
```

Para implementar esta función, siga los siguientes pasos:

- Considere la imagen de entrada (**im**) como una imagen doble RGB.
- Defina  $N$  (tamaño del histograma), inicialmente igual a 32. NO use "32" en la parte restante del código.
- Calcule histogramas con  $N$  bins para los tres planos de color (use **imhist**).
- Normalice los histogramas.
- Defina tres nuevos vectores, iguales a los valores de los histogramas desde  $3 \cdot (N/4)$  a  $N$  (último cuartil).
- Los resultados de **imhist** son vectores de columna, de modo que los vectores reducidos en el paso anterior deben ser tres  $(N / 4) \times 1$  vectores de columna. Unifíquelos en una sola matriz  $(N / 4) \times 3$ .
- Haga que la matriz sea igual a la variable de salida: **feat**.

Tenga en cuenta que en este caso hemos convertido el vector de características en una "matriz de características". Esto no debería ser un problema.

Una vez que la función **ExtractFeatures** está funcionando, podemos hacer una función que tome como entrada el nombre del archivo, aplique el preprocesamiento y extraiga las características. El código debería ser así:



```

1  function CMatrix = ExtractFromFile(filename)
2  %
3  % Extract features from file name.
4  im1 = ColorEnhance2(filename);
5  CMatrix = ExtractFeatures(im1);
6  end
7
8

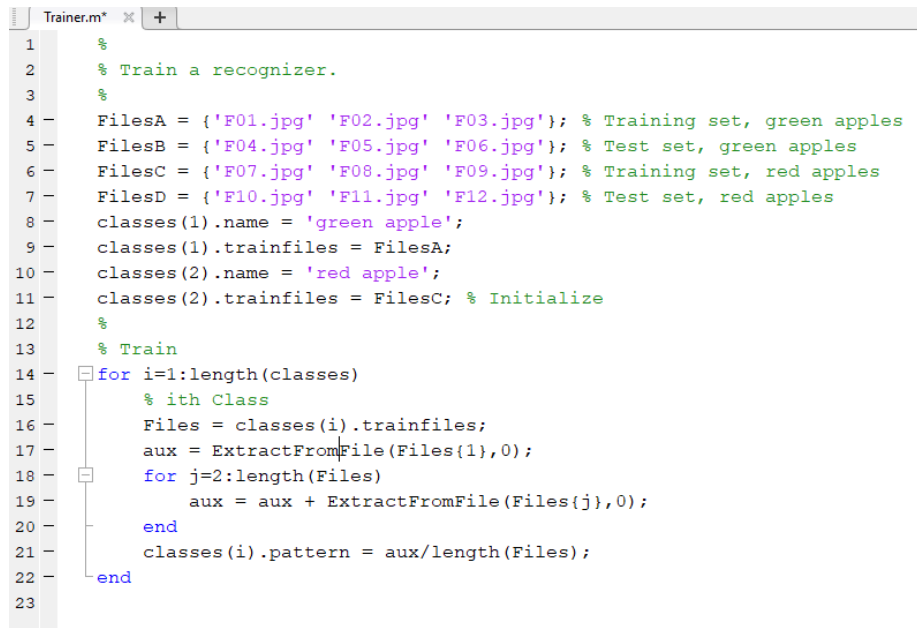
```

Figura 6. Preprocesamiento y extracción de características juntos.

## 7. Entrenamiento (cálculo de prototipos):

Una vez que podemos extraer características (en forma de matrices) de las imágenes, podemos calcular prototipos a partir de características de las imágenes de cada clase. Vamos a utilizar las primeras tres imágenes de cada clase como conjunto de entrenamiento (las tres imágenes restantes de cada conjunto serán el conjunto de prueba).

Para implementar esto, utilizaremos algunas capacidades de matlab que pueden ser nuevas para usted. Por esta razón, se le proporciona todo este código de entrenamiento:



```

1 %
2 % Train a recognizer.
3 %
4 FilesA = {'F01.jpg' 'F02.jpg' 'F03.jpg'}; % Training set, green apples
5 FilesB = {'F04.jpg' 'F05.jpg' 'F06.jpg'}; % Test set, green apples
6 FilesC = {'F07.jpg' 'F08.jpg' 'F09.jpg'}; % Training set, red apples
7 FilesD = {'F10.jpg' 'F11.jpg' 'F12.jpg'}; % Test set, red apples
8 classes(1).name = 'green apple';
9 classes(1).trainfiles = FilesA;
10 classes(2).name = 'red apple';
11 classes(2).trainfiles = FilesC; % Initialize
12 %
13 % Train
14 for i=1:length(classes)
15     % ith Class
16     Files = classes(i).trainfiles;
17     aux = ExtractFromFile(Files{1},0);
18     for j=2:length(Files)
19         aux = aux + ExtractFromFile(Files{j},0);
20     end
21     classes(i).pattern = aux/length(Files);
22 end
23

```

Figura 7. Código para el cálculo del prototipo (entrenamiento).

Vea que estamos usando dos estructuras de datos interesantes. Primero, las matrices de celdas (<https://es.mathworks.com/help/matlab/cell-arrays.html>) se usan para administrar los nombres de archivo. En segundo lugar, se utiliza una matriz de estructura para gestionar la información sobre las dos clases. Las estructuras en matlab (<https://es.mathworks.com/help/matlab/ref/struct.html>) son muy similares a las estructuras en C / C ++ o a los registros en Pascal.

## 8. Clasificación:

Ahora, debemos codificar una función para la clasificación. Use esta definición:

**función C = Recognizer(inputfile, clases)**

Donde **inputfile** es el nombre de un archivo de imagen de clase "desconocida" (obviamente, nosotros conocemos la clase pero matlab no sabe si es una manzana verde o roja ). **clases** debe ser la matriz de estructura creada en la sección anterior, que contiene todas las clases posibles junto con sus prototipos.

La función debe extraer características del archivo de entrada (use **ExtractFromFile**) y compararlas con todos los prototipos. Recuerde que llamamos "el patrón" a las características del archivo de entrada. Tenga en cuenta que estamos trabajando con dos clases, pero es mejor codificar pensando que tenemos **n** clases, **n = longitud (clases)** . Para comparar matrices (características) podemos definir un operador de distancia. Utilizaremos lo siguiente:

```

aux = abs (patrón-prototipo);
distancia = suma(aux (:));

```

El valor devuelto debe ser el índice de la clase ganadora (la que tiene una distancia mínima al patrón de entrada). Para el código de la sección 7, "manzana verde" es C = 1 y "manzana roja" es C = 2.

## 9. Prueba general (cálculo de la matriz de confusión):

Una vez que tengamos un reconocedor, debemos probarlo sistemáticamente. La matriz de confusión es una herramienta muy utilizada para probar reconocedores y, en general, sistemas de inteligencia artificial. Coloquialmente se construye ejecutando reconocimiento en todas las imágenes de prueba y sumando resultados. Más concretamente, comenzamos con una matriz  $n \times n$  llena de ceros ( $n$  es el número

de clases). La fila  $i$  es el resultado de probar todas las muestras de la clase  $i$ . Si una muestra produce el resultado  $j$ , el elemento  $(i, j)$  de la matriz se incrementa en una unidad. Por supuesto, la matriz ideal (sin errores) tiene solo elementos distintos de cero en su diagonal. Vea un ejemplo de matriz de confusión en la Figura 8.

True Positive (TP) : Observation is positive, and is predicted to be positive.  
 False Negative (FN) : Observation is positive, but is predicted negative.  
 True Negative (TN) : Observation is negative, and is predicted to be negative.  
 False Positive (FP) : Observation is negative, but is predicted positive.

n = 165	Predicted: No	Predicted: Yes	
Actual: No	Tn =50	FP=10	60
Actual: Yes	Fn=5	Tp=100	105
	55	110	

Figura 8. Ejemplo de matriz de confusión

Copie el código de la siguiente imagen para calcular la matriz de confusión para el "reconocedor de frutas".

```

ConfMatrix2.m x +
2 % Confusion matrix computation
3 % (with a given recognizer).
4 %
5 FilesA = {'F01.jpg' 'F02.jpg' 'F03.jpg'};
6 FilesB = {'F04.jpg' 'F05.jpg' 'F06.jpg'};
7 FilesC = {'F07.jpg' 'F08.jpg' 'F09.jpg'};
8 FilesD = {'F10.jpg' 'F11.jpg' 'F12.jpg'};
9 % A==> training class 1, B==> test class 1, C==> training
10 t(1).Files = FilesB;
11 t(2).Files = FilesD;
12 NF = length(classes);
13 ConfusionMatrix = zeros(NF,NF); % Initialize
14 %
15 % Compute matrix.
16 %
17 for i=1:NF
18     Files = t(i).Files;
19     for j=1:length(Files)
20         c = Recognizer(Files{j},classes);
21         ConfusionMatrix(i,c) = ConfusionMatrix(i,c)+1;
22     end
23 end
  
```

Figura 9. Cálculo de la matriz de confusión.

## 10. Material entregable:

Se le pedirá que suba los entregables de laboratorio a Faitic. La carga se realizará a través de un enlace que se activará durante el periodo apropiado. El material exacto a entregar será especificado por los profesores a través de Faitic con suficiente anterioridad, pero deberá estar basado en una función .mlx principal y una serie de funciones auxiliares que el alumno deberá entregar. El archivo .mlx deberá incluir

un breve texto que describa el desarrollo del código (esto puede estar dentro del código en forma de comentarios).

El alumno debe contestar también **obligatoriamente** a las siguientes cuestiones:

- Si se reduce N (número de BINS del histograma), ¿para qué valor experimentamos un rendimiento más bajo (deberíamos detectar esto usando la matriz de confusión)?
- ¿Se ve afectado el rendimiento si se elimina el preprocesamiento?
- ¿Cuál es la matriz de confusión si entrenamos sólo con dos imágenes por clase y dejamos el resto para la prueba?
- Pruebe otras funciones de distancia, por ejemplo:

```
1. distancia = norm(patrón-prototipo);  
2. distancia = norm(patrón-prototipo, 1);  
3. aux = (patrón-prototipo).^ 2; distancia = sum(aux (:));
```

**Trabajo OPCIONAL:** agreguen ustedes mismos otra clase; esto significa: tomar seis fotos de otro tipo de fruta. Introduzca las tres primeras fotos en la secuencia de entrenamiento y vuelva a calcular la matriz de confusión.