Visió per Computador

Homework 3: Chuletones

Huiwen Cao (**12F**)
Jordi Bru Carci (**12F**)
16/03/2023
2022-2023.Q2

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Tabla de contenidos

Introducción

Se nos proporciona un conjunto de imágenes a escala de gris de distintos chuletones. El objetivo de la práctica es detectar el porcentaje de grasa en unas chuletas.

Para ello, tenemos el script myscanfiles.m para poder leer todas las imágenes de un directorio.

Enunciado

- Entrada: Cap. Lectura directa de les 14 imatges del disc
- Sortides:
    - Visualització de les imatges binaritzades
    - Taula on aparegui el % de greix de cada llonza per als diferents mètodes.

Etapas

1. Seleccionar manualment una regió d'interès rectangular, de forma que s'elimini la part inferior de la imatge on reposa la llonza.

2. Implementar un petit programa que binaritzi les imatges a partir d'un llindar programable per l'usuari. Trobar el % de píxels de greix.
El % de greix cal trobar-lo respecte dels píxels totals de la llonza. Per tant, abans de fer el càlcul cal eliminar els píxels del fons.

3. Extreure l'histograma de les imatges. Trobar el llindar a ull a partir de l'histograma.
(Aquest histograma ens dona un interval de possibles llindars).

4. Trobar el llindar de forma automàtica. Podeu usar 'graythresh' de MATLAB.

5. Trobar el llindar utilitzant algun altre mètode. Implementeu AMB CODI PROPI, com a mínim, dos mètodes diferents.

Métodos para encontrar el threshold

Etapa 1. Eliminar la parte inferior de la imagen donde reposa la chuleta

En la etapa 1, tenemos que eliminar el fondo de la imagen, podemos observar que la parte inferior de la imagen donde reposa la chuleta tiene un textura diferente a la chuleta, se podría utilizar un threshold adecuado para segmentar esta región y eliminarla de la imagen. Al establecer un threshold, los objetos que tienen valores de píxeles más altos que el threshold se resaltan en la imagen binaria y se eliminan los objetos que tienen valores de píxeles más bajos que el threshold. El código convierte la imagen en una imagen binaria y encuentra todas las componentes conectadas. Luego, identifica el objeto más grande por su número de píxeles, lo aísla en una nueva imagen binaria y elimina todas las demás componentes conectadas de la imagen original.

Etapa 2. Threshold programable por el usuario

En la etapa 2, se está definiendo una variable llamada “threshold” que permite al usuario cambiar su valor. Una vez que se ha definido el valor de threshold, se aplica a todas las imágenes de la carpeta. Esto significa que el código va a procesar todas las imágenes en una carpeta y aplicar el valor de threshold definido por usuario para cada una de ellas.

Etapa 3. Mediante histogramas

En el ejercicio 3 nos pedía encontrar el threshold a ojo extrayendo el histograma de las imágenes. Como podemos ver en el código del anexo, hemos usado la función proporcionada por Matlab, imhist(). Si le pasamos como parámetro cada imágen con solo el chuletón podemos observar el siguiente output.

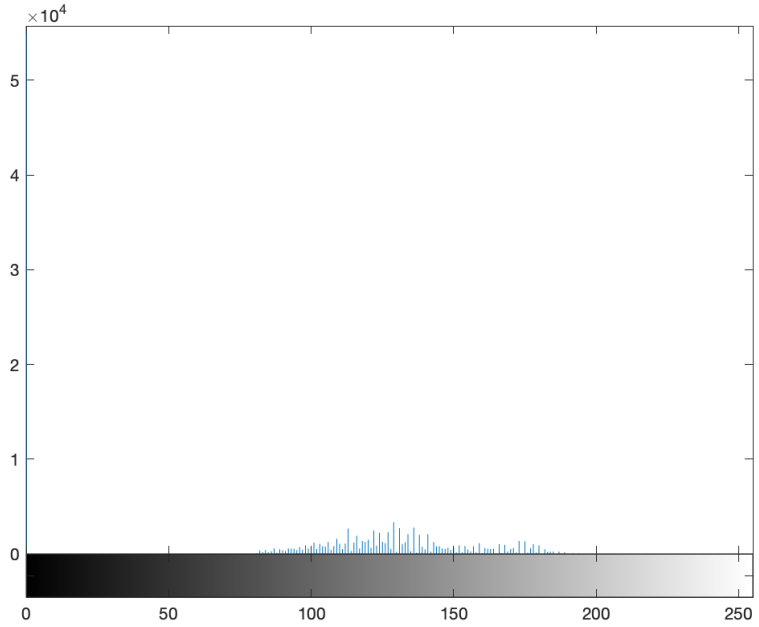


Figura 1. Output generado a partir de imhist(I)

Si consideramos que el eje X representa la intensidad del píxel y el eje Y el número de píxeles, podemos observar que los píxeles correspondientes a la grasa, identificados por su color, podrían empezar a partir de una intensidad de 150. Sin embargo, determinar el threshold adecuado a simple vista puede ser impreciso. Por lo tanto, podemos considerar un rango seguro de niveles entre 150 y 170.

Etapa 4. Automática: graythresh

En esta sección simplemente debemos usar la función de matlab graythresh() con la imágen del chuletón aislado para poder encontrar de forma automática el threshold óptimo. El nivel generado debemos multiplicarlo por 255 para poder ver un th con un valor de entre 0..255. Sinó nos da un valor entre 0..1. Además, como lo que queremos es buscar el threshold óptimo de la chuleta, debemos especificar I(I>0) para no tener en cuenta el fondo, el cual son todos píxeles a 0.

```
t_gray = 255*graythresh(I(I>0));
```

Etapa 5. Método 1: Otsu

Este método (ver otsu_thresholding.m en el Anexo) es llamado en el archivo porcentajes.m (Anexo) para calcular el threshold óptimo según el algoritmo de Otsu. De esta forma, una vez calculado, porcentajes.m (Anexo) calculará el % de grasa correspondiente para cada imagen.

Como breve explicación del procedimiento de la implementación, primero calculamos el histograma normalizado de la imagen. Luego se calculan las sumas acumulativas de las probabilidades de los niveles de gris y sus valores medios. Después, recorremos en bucle a través de todos los posibles valores de threshold y calculamos la varianza entre clases para cada threshold. El threshold que maximiza la varianza entre clases, lo seleccionaremos como el óptimo.

Etapa 5. Método 2: Iterativo

En este método (ver iterative.m en el Anexo) buscamos el threshold óptimo de forma iterativa. Primero cogemos un th medio entre el mínimo y el máximo. Luego dividimos la imagen en dos secciones, las que están por encima o por debajo del th establecido.

A partir de aquí, en el bucle, procedemos a calcular valores medios entre los grupos y haciendo medias entre ellos. De esta forma, llega un punto en el que la diferencia entre el th actual y el anterior es casi ínfima (establecemos 0.1 para valorarlo). Así pues, llegamos a la conclusión del threshold óptimo.

Anexo

Porcentajes.m

Este método es llamado en el live script principal recibiendo como parámetros todo lo necesario para poder calcular los porcentajes de los ejercicios 2, 4 y de los dos métodos del ejercicio 5.

Primero de todo, realiza las operaciones necesarias para quitar la mesa que sujeta el chuletón de cada imagen (etapa 1). Una vez logrado, procede a calcular los porcentajes de cada uno de los métodos mencionados.

```
function [p_u, p_gray, p_o, p_i] = porcentajes(I, threshold_user, file)
    %Etapa 1.Seleccionar manualment una regió d'interès rectangular,
    % de forma que s'elimini la part inferior de la imatge on reposa la llonza.
    BW = I > 100;
    C = bwconncomp(BW);
    CBW = BW;
    np = cellfun(@numel, C.PixelIdxList);
    [valormaxim, posicio] = max(np);

    for i=1:length(np)
        if i ~= posicio
            CBW(C.PixelIdxList{i}) = 0;
        end
    end
    I(CBW==0) = 0;

    %Etapa 2. Implementar un petit programa que binaritzi les imatges a partir d'un llindar
    %programable per l'usuari. Trobar el % de píxels de greix.
    h = imhist(I);
    hacum = cumsum(h);

    %Calcular el percentatge amb el threshold definit per usuari
    p_u = ( (hacum(256)-hacum(threshold_user)) / valormaxim ) * 100;

    %Etapa 4. Trobar el llindar de forma automàtica. Podeu usar graythresh de MATLAB.
    %Calcular el percentatge amb 'graythresh'
    t_gray = 255*graythresh(I(I>0));
    p_gray = ( (hacum(256)-hacum(t_gray)) / valormaxim ) * 100;

    %Etapa 5. Trobar el llindar utilitzant algun altre mètode. Implementeu AMB CODI PROPI,
    % com a mínim, dos mètodes diferents.
    %Calcular el threshold amb el mètode otsu
    t_o = otsu_thresholding(I(I > 0));
    p_o = ( (hacum(256)-hacum(t_o)) / valormaxim ) * 100;

    %Calcular el threshold amb el mètode iterative
    t_i = iterative(I(I > 0));
    p_i = ((hacum(256)-hacum(int32(t_i))) / valormaxim ) * 100;

    figure
    tlo = tiledlayout(3,3);
    %Imatge original
    nexttile
    imshow(I)
    title(['Imatge original'])
    %Threshold user
    nexttile
    imshow(I > threshold_user)
    title(['Threshold user'])
    %Graythresh
```

```
    nexttile
    imshow(I > t_gray)
    title(['Graythresh'])
    %Otsu
    nexttile
    imshow(I > t_o)
    title(['Otsu'])
    %Iterative
    nexttile
    imshow(I > t_i)
    title(['Iterative'])

    title(tlo,file)
end
```

otsu_thresholding.m

```
function threshold = otsu_thresholding(image)

% Extraiem l'histograma de la imatge
histogram = imhist(image);

% Normalitzem l'histograma per poder conseguir les seves probabilitats
histogram = histogram / sum(histogram);

% Calculem la suma acumulada de les probabilitats i les mitjanes.
cumulative_sum = cumsum(histogram);
mean_values = cumsum(histogram .* (1:numel(histogram))');

% Inicialitzem les variables per començar a tractar els llindars i la màxima variació
threshold = 0;
max_variance = 0;

% Bucle per tots els possibles llindars i calculant la variança
for t = 1:numel(histogram)
    % Calcula les probabilitats i mitjanes de les dues classes
    w0 = cumulative_sum(t);
    w1 = 1 - w0;
    u0 = mean_values(t) / w0;
    u1 = (mean_values(end) - mean_values(t)) / w1;

    % Calcula la variança entre les dues classes
    variance = w0 * w1 * (u0 - u1)^2;

    % Actualitzem el llindar i la màxima variança si és necessari
    if variance > max_variance
        threshold = t;
        max_variance = variance;
    end
end
end
```

iterative.m

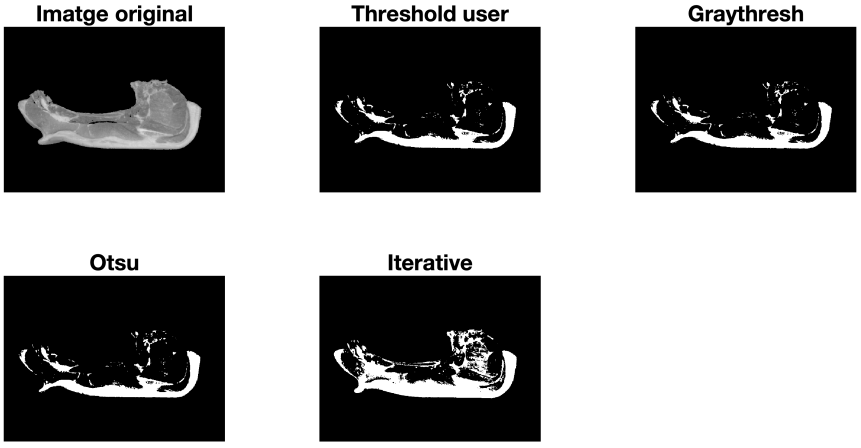
```
function T = iterative(image)
% La imatge es divideix per T (th mitjà). Per poder dividir la imatge en dos grups:
% G1 consisteix en tots els píxels amb un valor de gris major que T
% G2 consisteix en tots els píxels amb un valor de gris menor o igual que T
    T = 0.5*(min(image(:)) + max(image(:)));
    done = false;
    while ~done
        g = image > T;
        % Calcula els valors mitjos de la escala de grisos m1 i m2 en les zones G1 i G2
        Inext = 0.5*(mean(image(g))+mean(image(~g)));
        done = abs(T-Inext)<0.1;
        % Calcular el nou valor de llindar
        T = Inext;
    end
end
```

Código, imágenes binarizadas y tabla final

El código principal y los output con las imágenes y la tabla final, se muestran en el pdf generado del proyecto. El cual, como podemos observar en las siguientes páginas, recorre cada imágen llamando a los métodos ya documentados.

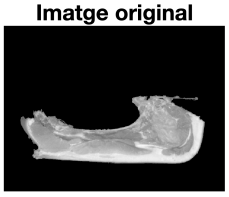
```
% script per obrir tots els arxius bmp d'una carpeta
clear all
close all
a = dir('./Chuletons/*.bmp');
nf = size(a);
T = table;
for i = 1:nf
filename = horzcat(a(i).folder,'/',a(i).name);
I = imread(filename);
%Obtenir el nom de cada imatge
name = split(a(i).name, ".");
file = name(1,:);

% el threshold programable per a l'usuari
threshold_user = 160;
[p_u, p_gray, p_o, p_i] = porcentajes(I, threshold_user, file);
structT(i, 1).name = file;
structT(i, 1).p_user= p_u;
structT(i, 1).p_graythresh = p_gray;
structT(i, 1).p_otsu = p_o;
structT(i, 1).p_iterative = p_i;
end
```

F1011flb

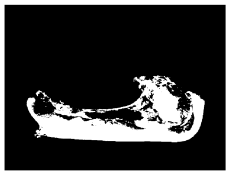


F1019flb

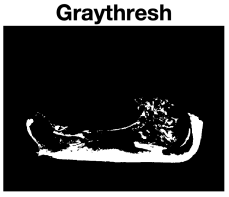
Imatge original



Threshold user

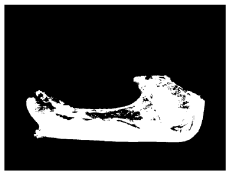


Graythresh



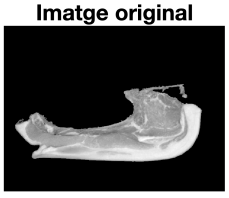
Otsu

Iterative

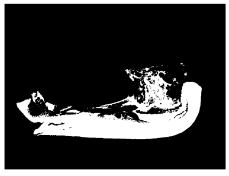


F1031flb

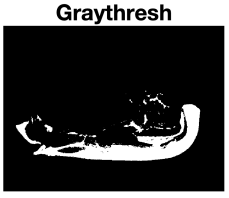
Imatge original



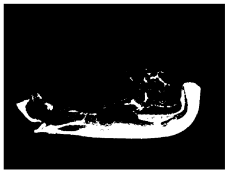
Threshold user



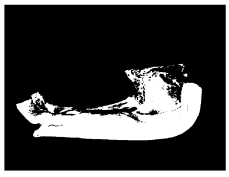
Graythresh



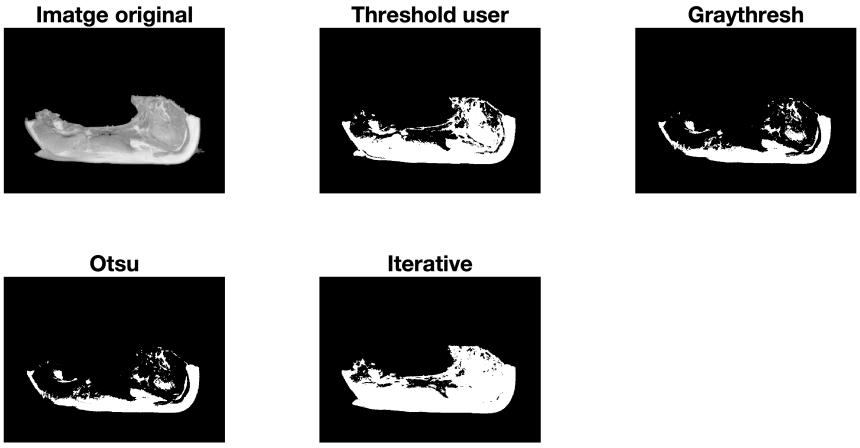
Otsu



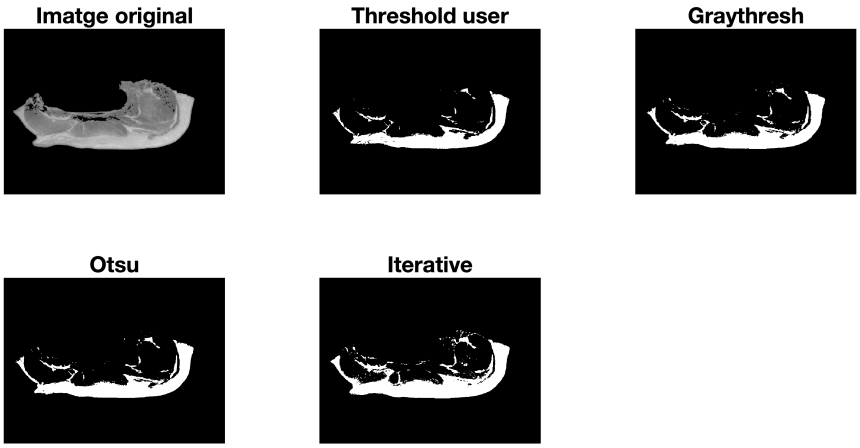
Iterative



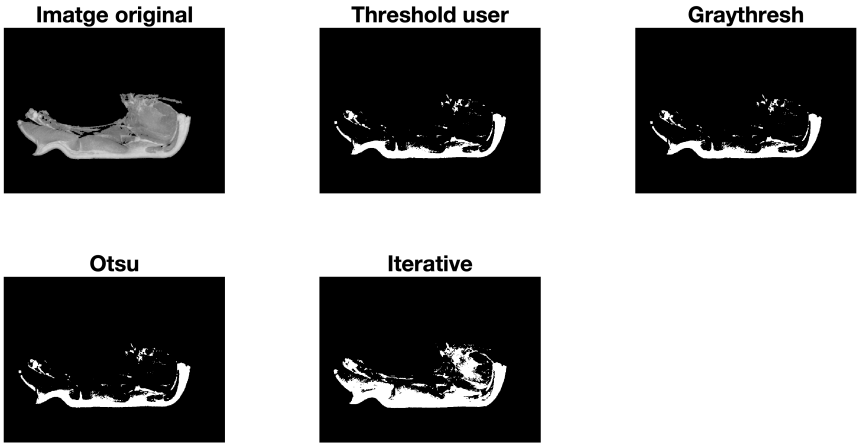
F1051flb



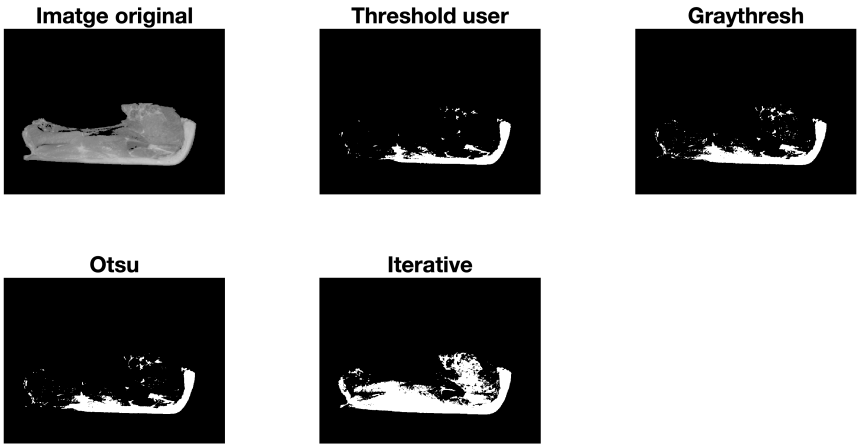
F1053flb



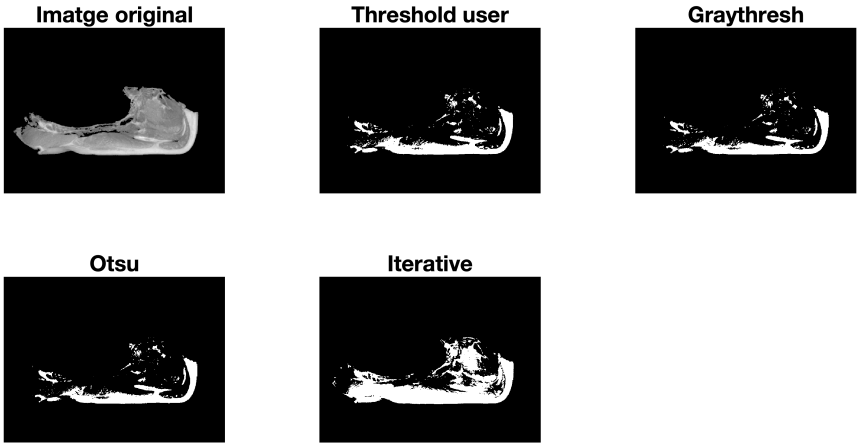
F1059flb



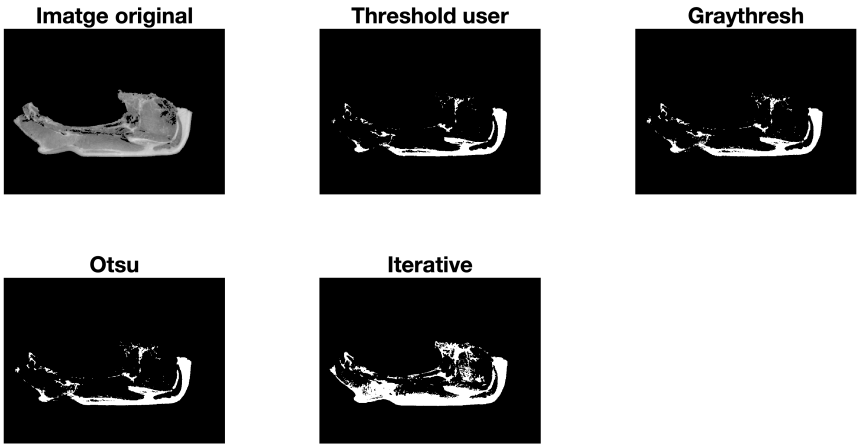
F1064flb



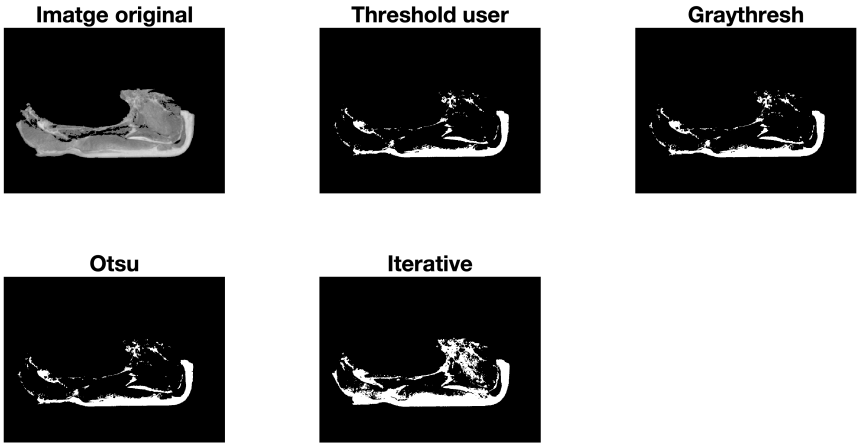
F1079flb



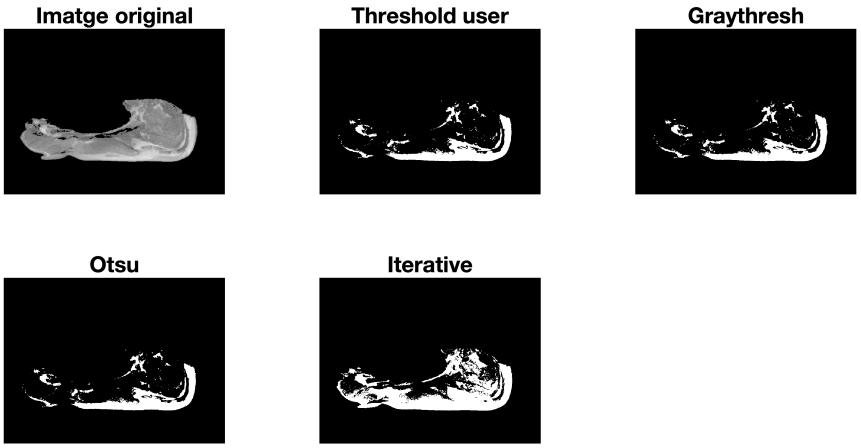
F1083flb



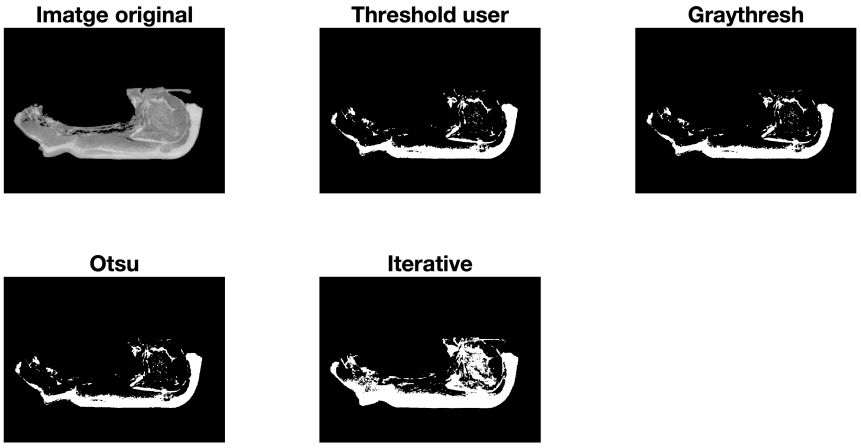
F1096flb



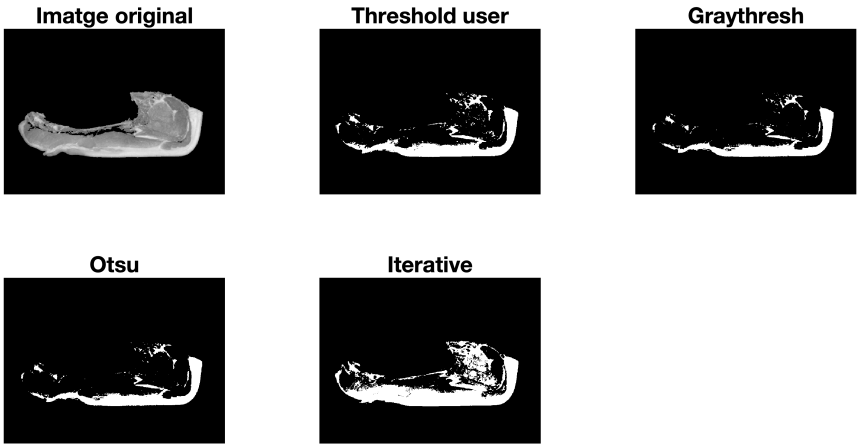
F1097flb



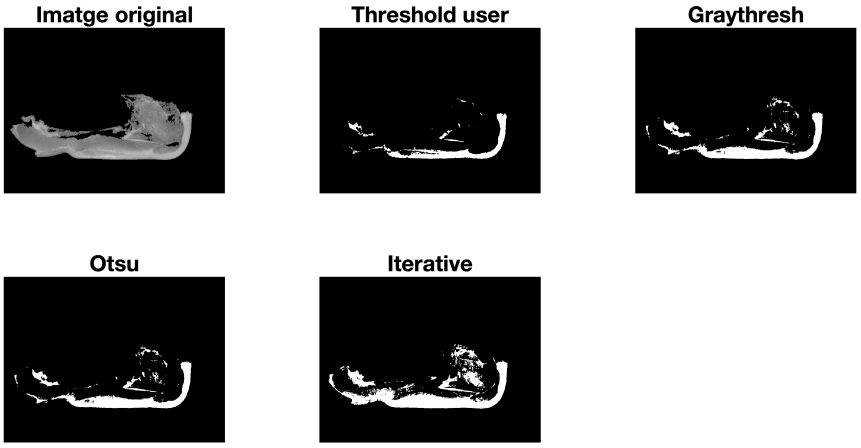
F1101flb



F1102flb



F1103flb




```
Tnew = struct2table(structT)
```

Tnew = 14×5 table

	name	p_user	p_graythresh	p_otsu	p_iterative
1	'F1011flb'	28.4227	29.7183	28.4227	54.7618
2	'F1019flb'	67.8305	34.5162	34.2160	82.2469
3	'F1031flb'	67.8820	38.3547	38.0107	78.6697
4	'F1051flb'	67.8208	37.1134	36.8546	83.1088
5	'F1053flb'	32.1113	37.2255	36.8788	46.2748
6	'F1059flb'	28.9057	28.9057	28.0761	57.1491
7	'F1064flb'	18.8602	25.9257	25.7917	60.7896
8	'F1079flb'	31.7835	28.1110	27.8793	60.5966
9	'F1083flb'	22.6367	26.7049	25.4567	49.4703
10	'F1096flb'	25.8198	27.1060	26.8759	53.2190
11	'F1097flb'	29.3299	29.3299	28.4552	62.2759
12	'F1101flb'	33.6112	33.6112	32.6690	59.7255
13	'F1102flb'	35.2532	28.4939	27.7142	60.1734
14	'F1103flb'	17.5919	29.7312	28.7026	47.2125