

# SISTEMA 1:

## Sistema de detección de piel.

Pol Buitrago

### a. Presentación del problema

En este proyecto, nos enfrentamos al desafío de desarrollar un sistema capaz de reconocer áreas de piel en imágenes que representan gestos con las manos. El objetivo final es contar el número de dedos mostrados en estas imágenes, lo que permitirá reconocer números del 0 al 5 en lenguaje de señas. Para abordar este problema, contamos con un conjunto de datos dividido en dos subconjuntos principales: el conjunto de entrenamiento (training dataset) y el conjunto de validación (validation dataset).

Cada uno de estos subconjuntos se encuentra organizado en dos subcarpetas. La primera subcarpeta contiene imágenes que se utilizarán para entrenar nuestro sistema. Estas imágenes representan gestos de manos, en las cuales a veces también se muestra parte del brazo o, en ocasiones, de otras partes del cuerpo tales como el cuello. Por otro lado, la segunda subcarpeta del conjunto contiene las máscaras ideales que indican las áreas de piel de las imágenes.

El segundo subconjunto, el conjunto de validación, presenta una estructura similar. En su primera subcarpeta, encontramos imágenes similares a las del conjunto de entrenamiento, pero estas se utilizarán para evaluar el rendimiento de nuestro sistema después de su desarrollo. La segunda subcarpeta en el conjunto de validación contiene las máscaras ideales que definen las áreas de piel en estas imágenes de validación. No obstante, estas máscaras ideales no pueden utilizarse para entrenar a nuestro conjunto, sino meramente para evaluar la eficiencia de nuestro sistema una vez desarrollado.

Nuestro objetivo principal es desarrollar algoritmos y técnicas que nos permitan identificar de manera efectiva y precisa las áreas de piel en las imágenes de gestos de manos. Este proceso se llevará a cabo en dos fases principales: en la primera fase, se construirá un modelo de piel basado en el color, y en la segunda fase, se contará el número de dedos detectados en las áreas de piel identificadas.

El resultado final esperado es un sistema que pueda reconocer números del 0 al 5 en lenguaje de señas a partir de las imágenes proporcionadas. Para evaluar la calidad de nuestro sistema, utilizaremos métricas como el

F-Score y el tiempo de cálculo. También exploraremos cómo diferentes parámetros y técnicas afectan el rendimiento del sistema y presentaremos recomendaciones para futuras mejoras.

En esta primera parte como bien se ha comentado nos centraremos en la primera parte del sistema, el sistema de detección de piel.

## **Sistema de detección de pieles**

La tarea, en su esencia, consiste en una estructura capaz de detectar la presencia de piel en imágenes, ya sea que dichos elementos de piel estén presentes o no en las mismas. Pues el objetivo de un sistema ingenioso como debe serlo este es identificar su presencia de la manera más precisa y exacta posible, con el propósito de reducir al mínimo tanto los falsos positivos como los falsos negativos.

En un inicio, nos enfrentamos a una escasez de conocimiento y, por ende, hemos tenido que investigar y evaluar las posibles técnicas para detectar píxeles de piel en imágenes. Esto implica primeramente trabajar en el espacio de color que mejor se adapte a la percepción de la piel humana, pues nuestra meta es encontrar un modelo cromático que facilite al máximo la detección de píxeles de piel, uno que funcione efectivamente para todos los tipos de piel, independientemente de la raza.

En consecuencia, nos basaremos en el color de la piel como se nos sugirió en la propuesta. La crominancia asociada a la piel humana es una característica bastante constante, con una variación mínima entre diferentes individuos, incluso de diferentes razas, siempre y cuando se tenga la misma iluminación en la escena.<sup>1</sup>

No obstante, existen otros factores que influirán en la eficacia de nuestro detector de piel. Estos factores incluyen, por ejemplo, el tamaño de la base de datos utilizada, es decir, la cantidad de muestras con las que entrenamos nuestro modelo o la cantidad de detalle del histograma empleado y la forma de analizarlo.

A fin de cuentas, debemos ser capaces de distinguir y separar los píxeles de interés de los que no lo son, lo que primeramente nos lleva a la elección crucial del espacio de color, que abordaremos en el siguiente punto.

En otras consideraciones, debemos también contemplar posibles procesos de postprocesamiento, como operaciones morfológicas, filtrado, detección de contornos, relleno de regiones y eliminación de sombras. Todas estas cuestiones han sido abordadas y minuciosamente analizadas en pos de mejorar la eficacia y el rendimiento de nuestro sistema.

# ÍNDICE

---

<b>a. Presentación del problema</b> .....	1
Sistema de detección de pieles .....	2
<b>b. Razonamiento y justificación de las elecciones hechas</b> .....	4
Elección del espacio de color .....	4
<b>function</b> algo1(points, dataset_type) .....	7
<b>function</b> skin_mask = algo2(input_image, points) .....	12
<b>function</b> algo3Simple(points, dataset_type) .....	14
<b>function</b> FScore = algo4(dataset_type) .....	17
optimizarUmbral.m.....	20
1. Múltiples regiones .....	24
2. Eliminar componentes pequeñas.....	27
3. Apertura y umbral de apertura .....	31
4. Relleno de regiones .....	40
5. Suavizado por apertura.....	42
6. Reducción de sombras .....	45
<b>function</b> algo3(points_matrix, dataset_type, umbralApertura, processingOptions)..	52
<b>function</b> FScore = algo4Graphics(dataset_type, createPlot).....	57
<b>c. Presentación de los resultados obtenidos:</b> .....	75
<b>F-Score en el conjunto de validación y en el conjunto entregado para la evaluación final (criterio 1)</b> .....	75
Training Dataset .....	76
Validation Dataset.....	79
Test Dataset .....	83
<b>Tiempo de cálculo<sup>1*</sup> en obtener resultados para una imagen (criterio 2)</b> ....	87
<b>d. Análisis de los resultados</b> .....	89
Errores Notables .....	91
Mejoras Implementadas .....	94
Estudio de la Sensibilidad de los Parámetros.....	94
Resultados de Tiempo de Ejecución .....	96
Resultados de F-Score .....	96
Interfaz Accesible .....	96
<b>e. Finalmente, presentación de ideas para mejorar el sistema.</b> .....	97
Conjunto de Entrenamiento Mayor.....	97
Regiones Elípticas.....	98
Distinción de Partes del Cuerpo.....	99
Detección en Tiempo Real.....	101
Mejora de la Interfaz de Usuario .....	103
Implementación de Redes Neuronales.....	104
Optimización del Rendimiento .....	106
Aplicaciones en Detección Médica.....	107
Histogramas de 128 Niveles.....	109
<b>Nota del autor</b> .....	123
<b>Referencias</b> .....	124

## **b. Razonamiento y justificación de las elecciones hechas**

Resulta ser que la piel en las imágenes es un elemento prácticamente invariable ante cambios en la luminancia. En otras palabras, la apariencia de la piel tiende a mantenerse consistente a pesar de las diferencias en la intensidad de la luz presente en las fotografías. Esta característica de estabilidad en la apariencia de la piel frente a las alteraciones en la luminosidad nos sugiere la conveniencia de emplear un espacio de color que exhiba una mínima susceptibilidad a estos cambios lumínicos. En consecuencia, elegir un espacio de color que conserve su coherencia ante las fluctuaciones de la iluminación se revela como la elección más adecuada.

### **Elección del espacio de color**

El modelo RGB, el más común y sencillo de todos, resultó ser, valga la redundancia, demasiado básico. En el caso de dos imágenes con un mismo sujeto (y, por ende, la misma piel) pero con niveles de iluminación diferentes, la distribución de píxeles en el histograma RGB varía de manera significativa.<sup>2</sup> Por lo tanto, se descartó la elección de RGB como espacio de color. Y, en su lugar, hemos optado por utilizar el modelo YCbCr, un espacio de color definido por “Y” representa la luminancia (brillo), y por “Cb” y “Cr”, que representan las componentes de crominancia (información de color). Para nuestra aplicación concreta, excluyendo la primera dimensión, pues esta (Y) representa la luminosidad y como ya hemos visto no es relevante a la hora de caracterizar los píxeles de piel de las personas.

Si cogemos una imagen al azar de la base de datos y la separamos en los tres canales de YCbCr, podemos observar cómo esta separación resalta diferentes aspectos clave de la imagen. A continuación, presentamos un ejemplo:

**Imagen 1: Canal Y (Luminancia)**



El canal Y se centra en la luminancia de la imagen, es decir, en el brillo de la mano. Como se puede apreciar en la imagen de muestra, la información de luminancia se presenta en escala de grises y no aporta información significativa sobre el color de la piel.

**Imagen 2: Canal Cb (Crominancia Azul)**



El canal Cb se enfoca en la información de crominancia azul. En esta imagen, podemos observar cómo las variaciones de color de la piel se destacan, lo que lo convierte en un canal adecuado para detectar la piel, ya que se pueden apreciar claramente las diferencias de color con respecto a otras partes de la imagen.

**Imagen 3: Canal Cr (Crominancia Roja)**



El canal Cr, por su parte, se enfoca en la información de crominancia roja. Al igual que en el canal Cb, se resaltan las variaciones de color de la piel en relación con el fondo o las áreas que no son piel. Esto demuestra cómo las componentes de crominancia Cb y Cr son valiosas para identificar la piel en las imágenes.

Estas imágenes muestran claramente cómo la separación en los canales de YCbCr permite resaltar las características de la piel, que son esenciales para la detección precisa de áreas de piel en las imágenes de gestos de manos. La elección de este espacio de color se basa en la capacidad de sus componentes de crominancia para captar estas variaciones de color que son características de la piel, independientemente de las diferencias en la iluminación. En la siguiente sección, exploraremos más a fondo cómo se utilizan estos canales para la detección de piel y los resultados obtenidos.

Para crear el sistema que se nos pide hemos diseñado un conjunto de scripts de MATLAB para diseñar nuestro modelo de detección de pieles en código. Empezando por la primera función la hemos llamado “**algo1**” y su función es generar un histograma 3D de los píxeles de piel en el espacio de color YCbCr de un cierto conjunto de imágenes y máscaras. Además, le hemos añadido una funcionalidad extra que permite insertar unos puntos de control de entrada referentes a valores de Cb y Cr para así dibujar sobre el histograma 3D la región deseada. Esta funcionalidad será de gran utilidad posteriormente, cuando tratemos de delimitar las regiones donde se hayan los píxeles de piel.

La función toma como entrada un vector de 4 puntos de control “[Cb\_min, Cb\_max, Cr\_min, Cr\_max]” que define los límites para identificar regiones con los píxeles de piel en las imágenes. Además, la función tiene un parámetro de entrada con el cual se especifica el tipo de conjunto de datos, que puede ser "Training" o "Validation." No obstante, ese parámetro se mantiene **SIEMPRE** constante en Training pues nosotros no tenemos acceso a las máscaras ideales de Validation ni podemos basar nuestro sistema en información extraída de ellas. La funcionalidad de poder especificar el tipo de banco de datos como entrada solo se ha añadido para hacer el código más versátil y pode cambiar fácilmente el banco de datos si quisiéramos usar otro.

## function algo1(points, dataset\_type)

```
1 function algo1(points, dataset_type)
2 %-----
3 %
4 % algo1 - Generar un histograma 3D de los píxeles de piel en el
5 % espacio de color YCbCr.
6 %
7 %
8 %
9 % Entradas:
10 % - points: Vector de 4 puntos de control [Cb_min, Cb_max, Cr_min, Cr_max].
11 % - dataset_type: Tipo de conjunto de datos, "Training" o "Validation".
12 %
13 %
14 %
15 % Explicación de la función:
16 % La función algo1 analiza las imágenes del conjunto de entrenamiento o
17 % validación y genera un histograma 3D de las componentes de crominancia
18 % Cb y Cr. Los puntos de control especificados definen los límites para
19 % identificar los píxeles de piel en las imágenes. El histograma resalta
20 % las regiones de piel en el espacio de color YCbCr y muestra los límites
21 % definidos por los puntos de control.
22 %
23 %
24 %
25 % Ejemplo de uso:
26 % points = [Cb_min, Cb_max, Cr_min, Cr_max]; % Definir puntos de control
27 % algo1(points, 'Validation'); % Generar histograma para el conjunto de
28 % validación
29 %
30 %
31 %
32 % Salida:
33 % La función no tiene una salida explícita, pero genera un histograma 3D
34 % y lo muestra en la figura actual.
35 %
36 %
37 %
38 %
39 % Verificar si se proporcionaron los valores de puntos de control
40 if nargin < 1
41     error(['Debes proporcionar un vector de puntos de ' ...
42            'control [Cb_min, Cb_max, Cr_min, Cr_max].']);
43 end
44 %
45 % Asegurarse de que el vector de puntos tenga 4 elementos
46 if numel(points) ~= 4
47     error(['El vector de puntos de control debe contener ' ...
48            'exactamente 4 valores.']);
49 end
50 %
51 Cb_min = points(1);
52 Cb_max = points(2);
53 Cr_min = points(3);
54 Cr_max = points(4);
55 %
56 % Verificar si se proporcionó el directorio del conjunto de datos
57 if nargin < 2
58     % Valor predeterminado del directorio del conjunto de datos
59     dataset_type = 'Training'; % Directorio "Training-Dataset" por defecto
60 end
61 %
62 % Agregar "-Dataset" al nombre del directorio
63 dataset_dir = strcat(dataset_type, '-Dataset');
64 %
65 % Definimos el directorio de las imágenes de entrenamiento
66 % o validación y las máscaras ideales
67 img_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
68                  dataset_dir, 'Images');
69 mask_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
70                     dataset_dir, 'Masks-Ideal');
71 %
72 % Lista de imágenes en el directorio
73 image_files = dir(fullfile(img_dir, '*.jpg'));
74 %
75 % Inicializamos vectores para almacenar los datos de crominancia
76 vect_im_mask_cb = [];
77 vect_im_mask_cr = [];
78
```

```

79 % Recorremos cada imagen del directorio
80 for n = 1:length(image_files)
81     % Leemos la imagen
82     img = imread(fullfile(img_dir, image_files(n).name));
83
84     % Transformamos de RGB a YCbCr
85     img_ycbcr = rgb2ycbcr(img);
86
87     % Extraemos las componentes de crominancia Cb y Cr
88     img_cb = img_ycbcr(:,:,2);
89     img_cr = img_ycbcr(:,:,3);
90
91     % Leemos la máscara correspondiente
92     mask_name = strrep(image_files(n).name, 'jpg', 'bmp');
93     mask = imread(fullfile(mask_dir, mask_name));
94
95     % Aplicamos la máscara a las componentes Cb y Cr
96     img_mask_cb = img_cb(mask);
97     img_mask_cr = img_cr(mask);
98
99     % Concatenamos los vectores con datos de crominancia
100    vect_im_mask_cb = [vect_im_mask_cb; img_mask_cb];
101    vect_im_mask_cr = [vect_im_mask_cr; img_mask_cr];
102 end
103
104 figure;
105 % Creamos la matriz CbCr resultante
106 matriz_cbcr_final = [vect_im_mask_cb, vect_im_mask_cr];
107
108 % Creación y presentación del histograma en 3D
109 hist3(matriz_cbcr_final, 'Edges', {0:1:255, 0:1:255}, 'CDataMode', 'auto');
110 xlabel('Componente Cb');
111 ylabel('Componente Cr');
112 zlabel('Frecuencia');
113 title('Histograma de las Componentes de Crominancia Cb y Cr');
114
115 % Invertir los colores del colormap
116 colormap(flipud(colormap));
117 % Mostrar el gráfico
118 colorbar; % Agrega una barra de colores para mostrar
119     % la correspondencia de valores
120
121 % Dibujar líneas verticales y horizontales para marcar los límites
122 hold on;
123 line([Cb_min, Cb_min], [0, 255], 'Color', 'r', 'LineWidth', 2);
124 line([Cb_max, Cb_max], [0, 255], 'Color', 'r', 'LineWidth', 2);
125 line([0, 255], [Cr_min, Cr_min], 'Color', 'r', 'LineWidth', 2);
126 line([0, 255], [Cr_max, Cr_max], 'Color', 'r', 'LineWidth', 2);
127 hold off;
128
129 % Agregar etiquetas a las líneas para mostrar los valores de los límites
130 text(Cb_min, 10, 'Cb min', 'Color', 'r', 'FontSize', 12);
131 text(Cb_max, 10, 'Cb max', 'Color', 'r', 'FontSize', 12);
132 text(10, Cr_min, 'Cr min', 'Color', 'r', 'FontSize', 12);
133 text(10, Cr_max, 'Cr max', 'Color', 'r', 'FontSize', 12);
134 end

```

La función “algo1” cumple las siguientes tareas:

1. Comprueba si se proporcionan los valores de puntos de control y garantiza que el vector de puntos tenga exactamente 4 elementos.
2. Delimita visualmente la región formada por los puntos de control Cb y Cr proporcionados en el espacio de color YCbCr.
3. Selecciona el directorio del conjunto de datos, que puede ser "Training" o "Validation."
4. Lee las imágenes y máscaras ideales del directorio correspondiente.

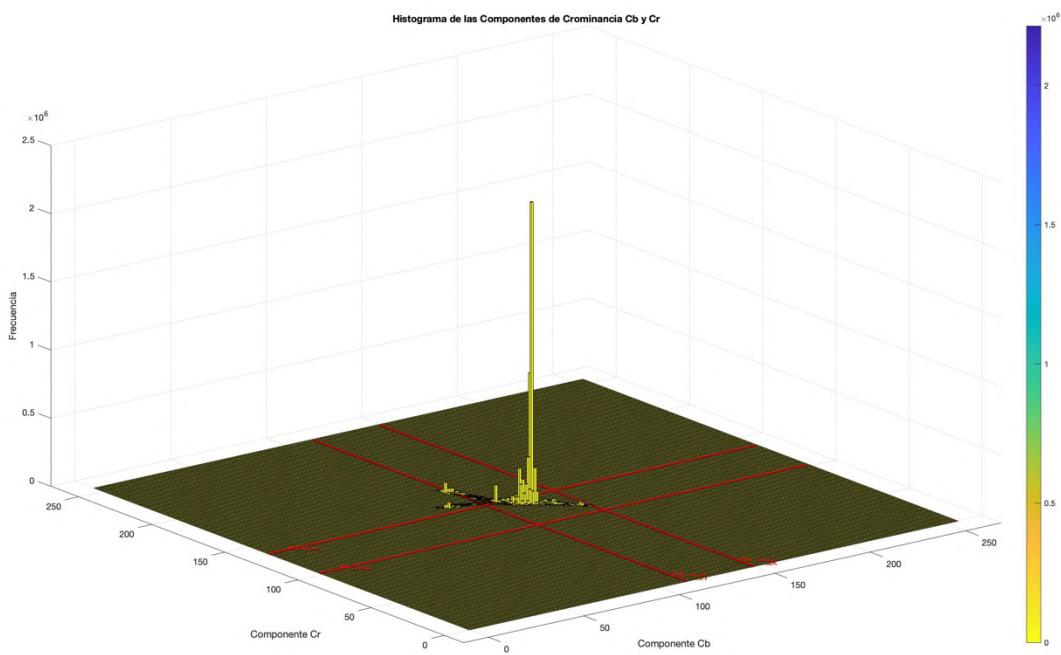
5. Transforma las imágenes de RGB a YCbCr para extraer las componentes Cb y Cr.
6. Aplica las máscaras ideales a las componentes Cb y Cr para obtener los datos de crominancia de los píxeles de piel.
7. Concatena los datos de crominancia Cb y Cr cada una en un solo vector.
8. Crea un histograma en 3D de las componentes de crominancia Cb y Cr y muestra los límites definidos por los puntos de control en el gráfico.
9. Invierte los colores del colormap para una mejor visualización.
10. Agrega etiquetas a las líneas que indican los valores que delimitan.

Esta función es esencial para comprender la distribución de las componentes de crominancia en el espacio de color YCbCr, lo que es crucial para la detección de piel. El algoritmo filtra las componentes Cb y Cr de las imágenes de testeo con las máscaras ideales para que así consigamos representar en el histograma solamente aquellos los valores Cb y Cr de aquellos píxeles que realmente son de piel, así pudiendo visualizar la distribución de los píxeles de piel en el espacio del espacio de color YCbCr.

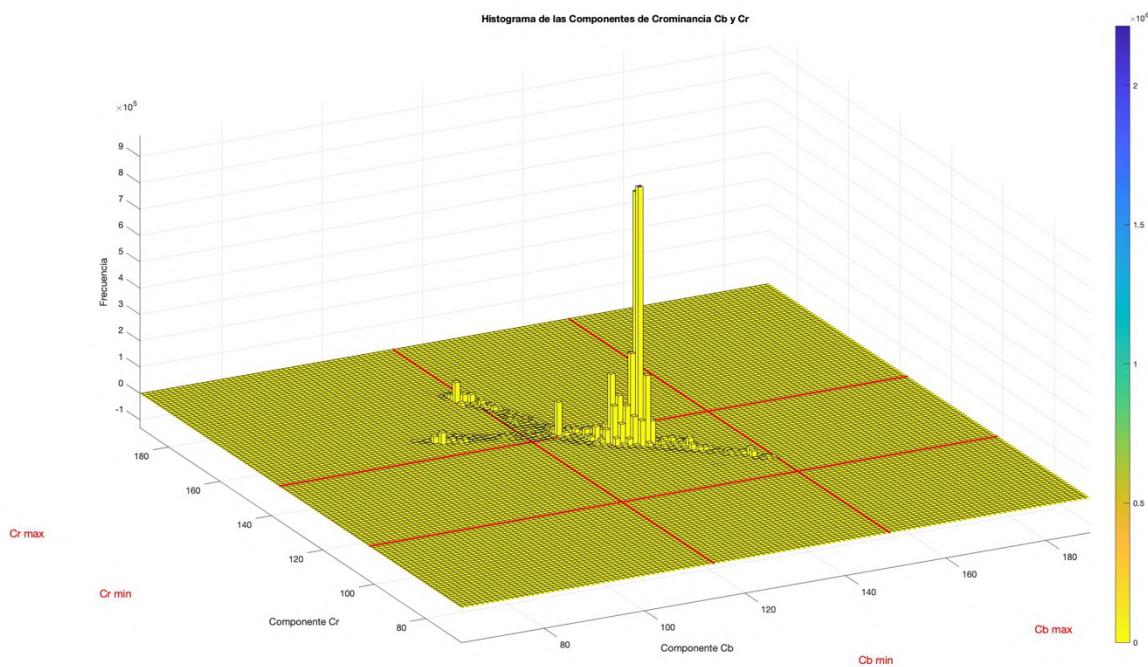
#### **Ejemplo de uso:**

Un ejemplo de uso típico de la función “algo1” es proporcionar una cierta región delimitada por los puntos de control “[Cb\_min, Cb\_max, Cr\_min, Cr\_max]” y especificar que se está trabajando con el conjunto de datos de entrenamiento. Luego, la función generará un histograma 3D que nos muestra la distribución de las componentes de crominancia Cb y Cr y podremos observar que región delimitan los puntos que hemos proporcionado.

A continuación, se proporciona un ejemplo de la gráfica obtenida con la función algo1 en el conjunto de datos de entrenamiento. En este caso con los valores Cb\_min = 114, Cb\_max = 149, Cr\_min = 102 y Cr\_max = 137, que como veremos a continuación parecen delimitar la mayor parte de las regiones de píxeles de piel.



Y si aplicamos un zoom para visualizar mejor las zonas con mayor concentración:



Una vez obtenemos el histograma con los píxeles de piel podemos observar que la mayoría de ellos se centran en puntos concretos, mayoritariamente en la región delimitada por los puntos comentados anteriormente. No obstante, este criterio podría parecer poco preciso para delimitar regiones pues no trabajamos con valores parametrizados, sino que estamos tomando regiones por un criterio meramente visual de forma gráfica. La región delimitada puede optimizarse y ajustarse justo en los límites que nos proporcionen una detección más eficiente, pero para ello necesitamos un parámetro que nos establezca valores de eficacia de nuestra detección. El parámetro utilizado para ello será el F-Score o medida-F en función de la precisión y la exhaustividad.

Comentado esto, primero debemos establecer más algoritmos que nos permitan a partir de estas regiones obtener máscaras y entonces obtener el F-Score comparando con las máscaras ideales.

Por ello vamos a definir una nueva función llamada “algo2” que nos generará una máscara de la piel en una imagen dada en formato RGB. Esta función toma como entrada la imagen de interés y un vector de 4 puntos de control “[Cb\_min, Cb\_max, Cr\_min, Cr\_max]”, que define la región donde, en teoría, se hayan los rangos de color de piel en el espacio de color YCbCr. La función crea una máscara binaria que identifica los píxeles de piel en la imagen original según los rangos de color especificados por los puntos de control.

A continuación, veamos la función:

## function skin\_mask = algo2(input\_image, points)

```
1 function skin_mask = algo2(input_image, points)
2 %
3 %
4 % algo2 - Genera una máscara de piel en una imagen dada.
5 %
6 %
7 %
8 % Entradas:
9 % - input_image: Imagen de entrada en formato RGB.
10 % - points: Vector de 4 puntos de control [Cb_min, Cb_max, Cr_min, Cr_max].
11 %
12 %
13 %
14 % Explicación de la función:
15 % La función algo2 toma una imagen en formato RGB y un vector de 4 puntos de
16 % control que definen los rangos de color de piel en el espacio de color YCbCr.
17 % La función crea una máscara binaria que identifica los píxeles de piel en la
18 % imagen original según los rangos especificados por los puntos de control.
19 %
20 %
21 %
22 % Ejemplo de uso:
23 % input_image = imread('imagen.jpg'); % Carga la imagen en formato RGB
24 % points = [Cb_min, Cb_max, Cr_min, Cr_max]; % Define los puntos de control
25 % skin_mask = algo2(input_image, points); % Genera una máscara de piel
26 %
27 %
28 % Salida:
29 % - skin_mask: Máscara binaria que indica los píxeles de piel
30 %             (1: piel, 0: no piel).
31 %
32 %
33 %
34 %
35 % Verificar si se proporcionaron los valores de Cb_min, Cb_max, Cr_min y Cr_max
36 if nargin < 2
37     error(['Debes proporcionar un vector de puntos de control' ...
38            '[Cb_min, Cb_max, Cr_min, Cr_max].']);
39 end
40 %
41 % Asegurarse de que el vector de puntos tenga 4 elementos
42 if numel(points) ~= 4
43     error('El vector de puntos de control debe contener exactamente 4 valores.');
44 end
45 %
46 Cb_min = points(1);
47 Cb_max = points(2);
48 Cr_min = points(3);
49 Cr_max = points(4);
50 %
51 % Inicializar la máscara con ceros
52 skin_mask = zeros(size(input_image, 1), size(input_image, 2));
53 %
54 % Transformar la imagen a YCbCr
55 im_ycbcr = rgb2ycbcr(input_image);
56 %
57 % Crear máscara basada en los rangos de color de piel
58 skin_mask((im_ycbcr(:,:,2) >= Cb_min & im_ycbcr(:,:,2) <= Cb_max) & ...
59           (im_ycbcr(:,:,3) >= Cr_min & im_ycbcr(:,:,3) <= Cr_max)) = 1;
60 end
```

La función “algo2” realiza las siguientes operaciones:

1. Verifica si se proporcionan los valores de los puntos de control y garantiza que el vector de puntos contenga exactamente 4 elementos.
2. Inicializa una máscara binaria con ceros del mismo tamaño que la imagen de entrada.

3. Transforma la imagen de entrada de RGB a YCbCr para acceder a las componentes de crominancia Cb y Cr.
4. Crea una máscara que identifica los píxeles de piel en función de los rangos de color definidos por los puntos de control.
5. La máscara resultante contiene píxeles con valor 1 para indicar piel y 0 para indicar áreas que no son piel.

#### **Ejemplo de uso:**

Un ejemplo típico de uso de la función “algo2” implica cargar una imagen en formato RGB, definir los puntos de control “[Cb\_min, Cb\_max, Cr\_min, Cr\_max]” que especifican los rangos de color de piel y utilizar la función para generar una máscara de piel. La máscara binaria resultante se puede utilizar para identificar las áreas de piel en la imagen original.

#### **Salida:**

La función “algo2” devuelve la máscara de piel en forma de una matriz binaria. En esta máscara, los píxeles de piel se representan con un valor de 1, mientras que los píxeles que no son piel tienen un valor de 0.

Cabe comentar que para que la función cree las máscaras deseadas los límites de la región deben estar bien delimitados y corresponder a los píxeles del elemento o elementos de los que se quiere crear una máscara. Por lo que para que esa máscara corresponda a la mano en una imagen los límites deberán ser lo más precisos posibles correspondientes a los píxeles de la piel. Más adelante veremos que esa cuestión tiene un mayor grado de complejidad del que aparenta y que las regiones no son tan sencillas de delimitar.

Ahora una vez tenemos una función capaz de generar máscaras para una determinada imagen necesitamos definir una nueva función que permita, a partir de un uso reiterado de la función anterior, generar máscaras de detección de piel para todo un conjunto de imágenes.

A posteriori, una vez ya realizamos pruebas de eficiencia con el sistema de detección completo y funcional nos dimos cuenta de que la detección de piel y específicamente de manos podía optimizarse y mejorarse a partir de diferentes algoritmos y sistemas que fuimos aplicando y eso suponía la modificación de la función algo “algo3” para interferir, adaptar y mejorar las máscaras generadas. No obstante, la versión de algo3 que se va a mostrar a continuación, denominada como algo3Simple, es una versión anterior que solo mantiene el código base de la función, es decir, generar máscaras de detección de piel de un conjunto de imágenes, sin ninguna operación ni acción adicional. Todas las modificaciones serán explicadas y aplicadas más adelante en este informe, una vez el sistema esté completo y pueda cuantificarse su eficacia.

### function algo3Simple(points, dataset\_type)

```

1 function algo3Simple(points, dataset_type)
2 %-
3 %
4 % algo3Simple - Genera máscaras de detección de piel para un conjunto
5 % de imágenes.
6 %
7 %
8 %
9 % Entradas:
10 % - points: Vector de 4 puntos de control (Cb_min, Cb_max, Cr_min, Cr_max).
11 % - dataset_dir: Directorio del conjunto de datos ('Training' o 'Validation').
12 %
13 %-
14 %
15 % Explicación de la función:
16 % La función algo3 utiliza la función algo2 para generar máscaras de detección
17 % de piel para un conjunto de imágenes. El conjunto de imágenes se define
18 % mediante un directorio ("Images") dentro del directorio del conjunto de datos
19 % especificado. Las máscaras se generan y se escriben en un directorio llamado
20 % "Masks" con el mismo nombre que las imágenes procesadas.
21 %
22 %
23 %
24 % Ejemplo de uso:
25 % points = [120, 140, 120, 140]; % Define los puntos de control como un vector
26 % dataset_dir = 'Training'; % Especifica el directorio del conjunto de datos
27 % algo3(points, dataset_dir); % Llama a la función para generar máscaras.
28 %
29 %
30 %
31 %
32 % Verificar si se proporcionaron los valores de puntos de control
33 if nargin < 1
34 %error(['Debes proporcionar un vector de puntos de control ' ...
35 %      '[Cb_min, Cb_max, Cr_min, Cr_max].']);
36 end
37 %
38 % Asegurarse de que el vector de puntos tenga 4 elementos
39 if numel(points) ~= 4
40 %error('El vector de puntos de control debe contener exactamente 4 valores.');
41 end
42 %
43

```

```

44 % Verificar y ajustar el directorio del conjunto de datos
45 if nargin < 2
46     dataset_type = 'Training'; % Valor predeterminado: directorio "Training"
47 end
48
49 % Agregar sufijo "-Dataset" al directorio si es necesario
50 if ~contains(dataset_type, '-Dataset')
51     dataset_dir = strcat(dataset_type, '-Dataset');
52 end
53
54 % Directorios de entrada y salida
55 input_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
56                     dataset_dir, 'Images');
57 output_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
58                     dataset_dir, 'Masks');
59
60 % Comprobar y crear el directorio de salida si no existe
61 if ~exist(output_dir, 'dir')
62     mkdir(output_dir);
63 end
64
65 % Información de las imágenes en el directorio de entrada
66 lista_im = dir(fullfile(input_dir, '*.jpg'));
67
68 % Para cada imagen en el directorio de entrada
69 for n = 1:length(lista_im)
70     % Leer la imagen
71     imagen = imread(fullfile(input_dir, lista_im(n).name));
72
73     % Crear la máscara
74     mask = algo2(imagen, points);
75
76     % Construir el nombre de archivo de salida
77     [~, base_name, ~] = fileparts(lista_im(n).name);
78     output_filename = fullfile(output_dir, [base_name, '.bmp']);
79
80     % Guardar la máscara en el directorio de salida
81     imwrite(mask, output_filename);
82 end
83 end

```

La función “algo3Simple” como bien se ha comentado, genera máscaras de detección de piel para un conjunto de imágenes. La función toma como entrada un vector de 4 puntos de control “[Cb\_min, Cb\_max, Cr\_min, Cr\_max]” que define los rangos de color de piel en el espacio de color YCbCr. Además, se especifica el tipo de conjunto de datos con el que se quiere trabajar, que puede ser “Training” o “Validation”.

La función `algo3Simple` realiza las siguientes tareas:

1. Verifica si se proporcionan los valores de los puntos de control y garantiza que el vector de puntos contenga exactamente 4 elementos.
2. Verifica y ajusta el directorio del conjunto de datos, utilizando “Training-Dataset” por defecto si no se especifica el sufijo “-Dataset”.
3. Define los directorios de entrada y salida para las imágenes y las máscaras.
4. Comprueba si el directorio de salida existe y, de no ser así, lo crea.

5. Obtiene una lista de imágenes en el directorio de entrada.
6. Para cada imagen en el directorio de entrada, aplica la función "algo2" para generar una máscara de piel.
7. Guarda la máscara en el directorio de salida con el mismo nombre que la imagen original, pero con la extensión ".bmp."

**Ejemplo de uso:**

Un ejemplo típico de uso de la función "algo3Simple" implica definir los puntos de control "[Cb\_min, Cb\_max, Cr\_min, Cr\_max]" y especificar el tipo de conjunto de datos (por ejemplo, "Training"). Luego, la función generará máscaras de detección de piel para todas las imágenes en el directorio de imágenes correspondiente.

Y ahora, nos faltaría una última función que nos compare las máscaras que hemos creado con las máscaras ideales y así viéramos su similitud. El valor con el cual podremos cuantificar la eficacia de nuestro sistema es el F-Score, una medida objetivamente cuantitativa de medición sobre la eficacia de nuestro modelo de detección de pieles. La métrica del F-Score es un único valor entre 0 y 1, y el objetivo de cualquier diseñador de algoritmos de detección es que esta métrica esté lo más próxima posible a 1. Podemos hacer variar el parámetro de “Recall” y ver como varia el otro parámetro “Precision”. La meta es que en cuanto más “Recall” tengamos, mejor se haga la “Precision” o como mínimo no bajar en valor, ya que esto significaría que en cuanto más objetos detecta, peor hace una detección correcta.

La función “algo4” calcula el valor de F-score al comparar las máscaras generadas por nosotros con las máscaras ideales en un conjunto de imágenes. El F-score nos va a permitir evaluar la calidad de las máscaras generadas en la detección de piel.

### function FScore = algo4(dataset\_type)

```

1 function FScore = algo4(dataset_type)
2 %
3 %
4 % algo4 - Calcula el F-score comparando máscaras generadas y máscaras ideales.
5 %
6 %
7 %
8 % Entradas:
9 % - dataset_type: Tipo de conjunto de datos, "Training" o "Validation."
10 %
11 %
12 %
13 % Explicación de la función:
14 % La función algo4 compara las máscaras generadas por nosotros con las
15 % máscaras ideales para un conjunto de imágenes y calcula el F-score.
16 % El F-score es una métrica que combina la precisión (Precision) y la
17 % exhaustividad (Recall) para evaluar la calidad de las máscaras generadas
18 % en la detección de piel.
19 %
20 %
21 %
22 % Ejemplo de uso:
23 % FScore = algo4('Validation'); % Calcular el F-score para el conjunto
24 %                               de validación
25 %
26 %
27 %
28 % Salida:
29 % - FScore: Valor del F-score que indica la calidad de las máscaras generadas.
30 %
31 %
32 %
33 % Verificar si se proporcionó el tipo de conjunto de datos
34 if nargin < 1
35     % Valor predeterminado del tipo de conjunto de datos
36     dataset_type = 'Training'; % Directorio "Training-Dataset" por defecto
37 end
38

```

```

39 % Agregar "-Dataset" al nombre del directorio
40 dataset_dir = strcat(dataset_type, '-Dataset');
41
42 % Directorios de entrada
43 input_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
44                         dataset_dir, 'Images');
45 mask_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
46                         dataset_dir, 'Masks');
47 ideal_mask_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
48                         dataset_dir, 'Masks-Ideal');
49
50 % Información de las imágenes en el directorio de entrada
51 lista_im = dir(fullfile(input_dir, '*.jpg'));
52
53 % Variables para el cálculo de F-score
54 vect_TP = zeros(1, length(lista_im));
55 vect_T = zeros(1, length(lista_im));
56 vect_P = zeros(1, length(lista_im));
57
58 % Por cada imagen en el directorio de entrada
59 for n = 1:length(lista_im)
60     % Leer la máscara generada por nosotros y la máscara ideal
61     mask_nuestra = imread(fullfile(mask_dir, strrep(lista_im(n).name, ...
62                                 'jpg', 'bmp')));
63     mask_ideal = ~imread(fullfile(ideal_mask_dir, strrep(lista_im(n).name, ...
64                                 'jpg', 'bmp')));
65
66     % Calcular TP, T y P utilizando operaciones vectorizadas
67     TP = sum(mask_nuestra(:) & mask_ideal(:));
68     P = sum(mask_nuestra(:));
69     T = sum(mask_ideal(:));
70
71     % Almacenar los resultados en vectores
72     vect_TP(n) = TP;
73     vect_T(n) = T;
74     vect_P(n) = P;
75 end
76
77 % Calcular TP, T y P totales
78 total_TP = sum(vect_TP);
79 total_T = sum(vect_T);
80 total_P = sum(vect_P);
81
82 % Calcular Precisión y Recall
83 Precision = total_TP / total_P;
84 Recall = total_TP / total_T;
85
86 % Calcular F-Score
87 FScore = 2 * Precision * Recall / (Precision + Recall);
88 end

```

La función “algo4” realiza las siguientes tareas:

1. Verifica si se proporciona el tipo de conjunto de datos, utilizando "Training-Dataset" por defecto si no se especifica el sufijo "-Dataset."
2. Define los directorios de entrada para las imágenes, las máscaras generadas por nosotros y las máscaras ideales.
3. Obtiene una lista de imágenes en el directorio de entrada.
4. Inicializa variables para el cálculo de True Positives (TP), Total Positives (P) y Total Actual Positives (T).
5. Para cada imagen en el directorio de entrada, compara la máscara generada por nosotros con la máscara ideal y calcula TP, P y T.

6. Calcula los totales de TP, T y P para todas las imágenes.
7. Calcula la Precisión (Precision) y la Exhaustividad (Recall) en función de los totales.
8. Calcula el F-Score utilizando la fórmula estándar que combina la Precisión y la Exhaustividad.

**Ejemplo de uso:**

Un ejemplo típico de uso de la función “algo4” consiste en especificar el tipo de conjunto de datos (por ejemplo, “Validation”) y luego llamar a la función para calcular el F-score en función de las máscaras generadas previamente por nosotros en el directorio Masks y las máscaras ideales en el conjunto de imágenes.

**Salida:**

La función “algo4” devuelve el valor del F-score, que proporciona una evaluación cuantitativa de la calidad de las máscaras generadas en la detección de piel. El F-score es una métrica útil para comprender cómo las máscaras generadas se comparan con las máscaras ideales en términos de precisión y exhaustividad.

Y con estas 4 funciones ya podríamos definir un sistema, algo rudimentario, con el que crear máscaras de manos de las imágenes que se nos proporcionen y luego evaluar la eficacia de nuestro sistema a partir del valor de F-Score que obtengamos con el conjunto. No obstante, en este punto solo podemos elegir la región de forma gráfica, lo cual difícilmente nos de la región óptima, además de ser un procedimiento poco técnico. No obstante, sí sabemos gracias al histograma donde se concentran la mayoría de los píxeles de piel y además tenemos un sistema que nos cuantifica la calidad de nuestras máscaras por lo que podríamos diseñar un algoritmo que iterase las 4 componentes de Cb y Cr y adaptar la región a la que nos proporcione un mayor valor F-Score. Es por ello que hemos diseñado el algoritmo conocido como “optimizarUmbral.m” que realiza una búsqueda exhaustiva de los valores óptimos para los parámetros de umbral Cb\_min, Cb\_max, Cr\_min y Cr\_max en la detección de piel.

## optimizarUmbral.m

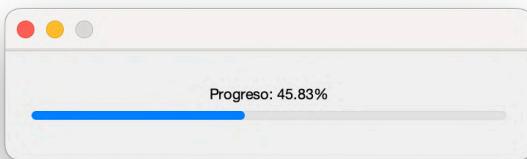
```
1 % Definir rangos de búsqueda para Cb_min, Cb_max, Cr_min y Cr_max
2 Cb_min_range = 108:1:130;
3 Cb_max_range = 140:1:160;
4 Cr_min_range = 90:1:150;
5 Cr_max_range = 120:1:150;
6
7 % Calcular el número total de iteraciones
8 total_iterations = length(Cb_min_range) * length(Cb_max_range) ...
9             * length(Cr_min_range) * length(Cr_max_range);
10
11 % Inicializar el contador de iteraciones completadas
12 completed_iterations = 0;
13
14 % Crear la barra de progreso
15 h = waitbar(0, 'Procesando...');
16
17 % Inicializar una matriz para almacenar los resultados del F-Score
18 FScore_matrix = zeros(length(Cb_min_range), length(Cb_max_range), ...
19                     length(Cr_min_range), length(Cr_max_range));
20
21 % Inicializar variables para almacenar los valores óptimos
22 optimal_Cb_min = 0;
23 optimal_Cb_max = 0;
24 optimal_Cr_min = 0;
25 optimal_Cr_max = 0;
26 max_FScore = -inf;
27
28 % Iterar a través de diferentes combinaciones de Cb_min, Cb_max, Cr_min y Cr_max
29 for i = 1:length(Cb_min_range)
30     for j = 1:length(Cb_max_range)
31         for k = 1:length(Cr_min_range)
32             for l = 1:length(Cr_max_range)
33                 % Obtener los valores de Cb_min, Cb_max, Cr_min y Cr_max
34                 Cb_min = Cb_min_range(i);
35                 Cb_max = Cb_max_range(j);
36                 Cr_min = Cr_min_range(k);
37                 Cr_max = Cr_max_range(l);
38
39                 points = [Cb_min, Cb_max, Cr_min, Cr_max];
40
41                 % Llamar a algo3 con los valores de Cb_min, Cb_max, Cr_min y Cr_max
42                 algo3Simple(points, 'Training');
43
44                 % Llamar a algo4 para obtener el FScore
45                 FScore = algo4();
46
47                 % Almacenar el FScore en la matriz
48                 FScore_matrix(i, j, k, l) = FScore;
49
50                 % Actualizar los valores óptimos si se encuentra un FScore mejor
51                 if FScore > max_FScore
52                     max_FScore = FScore;
53                     optimal_Cb_min = Cb_min;
54                     optimal_Cb_max = Cb_max;
55                     optimal_Cr_min = Cr_min;
56                     optimal_Cr_max = Cr_max;
57                 end
58
59                 % Incrementar el contador de iteraciones completadas
60                 completed_iterations = completed_iterations + 1;
61
62                 % Calcular el progreso en porcentaje
63                 progress_percent = (completed_iterations / total_iterations) * 100;
64
65                 % Actualizar la barra de progreso
66                 waitbar(progress_percent / 100, h, sprintf('Progreso: %.2f%', ...
67                         progress_percent));
68             end
69         end
70     end
71 end
72
73 % Cerrar la barra de progreso
74 close(h);
75
76 % Mostrar los valores óptimos
77 disp('Los valores óptimos son:');
78 disp(['Cb_min = ' num2str(optimal_Cb_min)]);
79 disp(['Cb_max = ' num2str(optimal_Cb_max)]);
80 disp(['Cr_min = ' num2str(optimal_Cr_min)]);
81 disp(['Cr_max = ' num2str(optimal_Cr_max)]);
82 disp(['El mejor F-Score encontrado es: ' num2str(max_FScore)]);
```

El procedimiento es el siguiente:

- Se definen rangos de búsqueda para Cb\_min, Cb\_max, Cr\_min y Cr\_max, y se define el paso entre valores, a ser posible un incremento lo menor posible, idealmente de 1 como en el ejemplo, para realizar una búsqueda exhaustiva.
- Se calcula el número total de iteraciones necesarias para cubrir todas las combinaciones de parámetros.
- Se inicializa un contador de iteraciones completadas y se crea una barra de progreso.



- Se inicializa una matriz para almacenar los resultados del F-Score.
- Se inicializan variables para almacenar los valores óptimos y el máximo F-Score encontrado.
- Se itera a través de diferentes combinaciones de parámetros, llamando a “algo3Simple” para generar máscaras de piel y a “algo4” para calcular el F-Score para cada combinación.
- Se almacenan los resultados en la matriz FScore\_matrix y se actualizan los valores óptimos si se encuentra un F-Score mejor.
- Se actualiza la barra de progreso con el progreso en porcentaje.



- Se cierra la barra de progreso después de completar todas las iteraciones.
- Se muestran en la consola los valores óptimos encontrados y el mejor F-Score.

Esta estrategia nos garantiza que se explore todo el espacio de búsqueda que le hayamos definido, lo que nos permite identificar los valores que maximizan el F-Score de manera precisa dentro de ese espacio. Además, se proporciona retroalimentación visual al usuario a través de una barra de progreso para mantenerlo informado sobre el progreso del proceso de búsqueda como hemos visto.

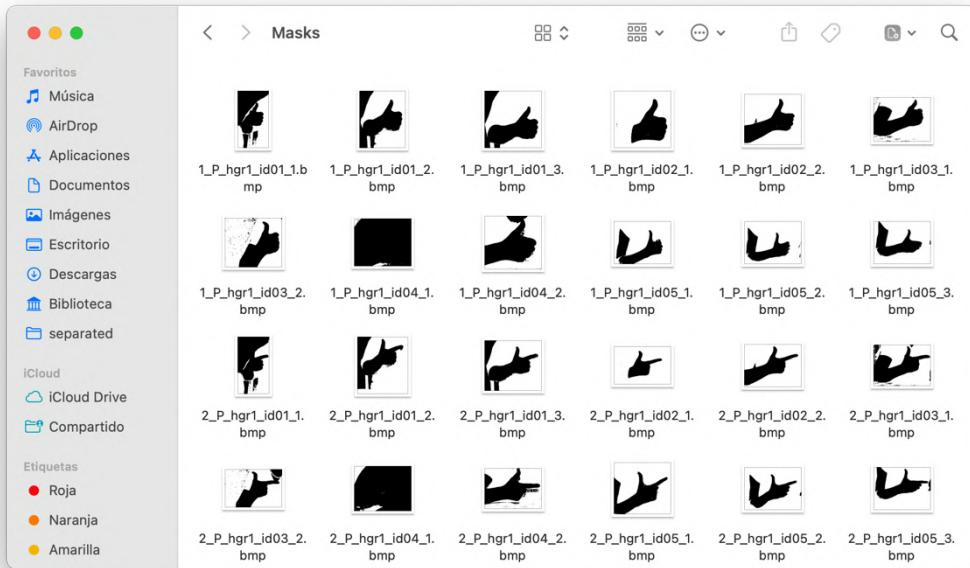
Gracias a este algoritmo logramos mejorar la eficiencia de nuestro sistema logrando obtener un F-Score de 0.92025 en el conjunto de entrenamiento.

```

>> optimizarUmbral
Los valores óptimos son:
Cb_min = 114
Cb_max = 149
Cr_min = 102
Cr_max = 137
El mejor F-Score encontrado es: 0.92025
fx >> |

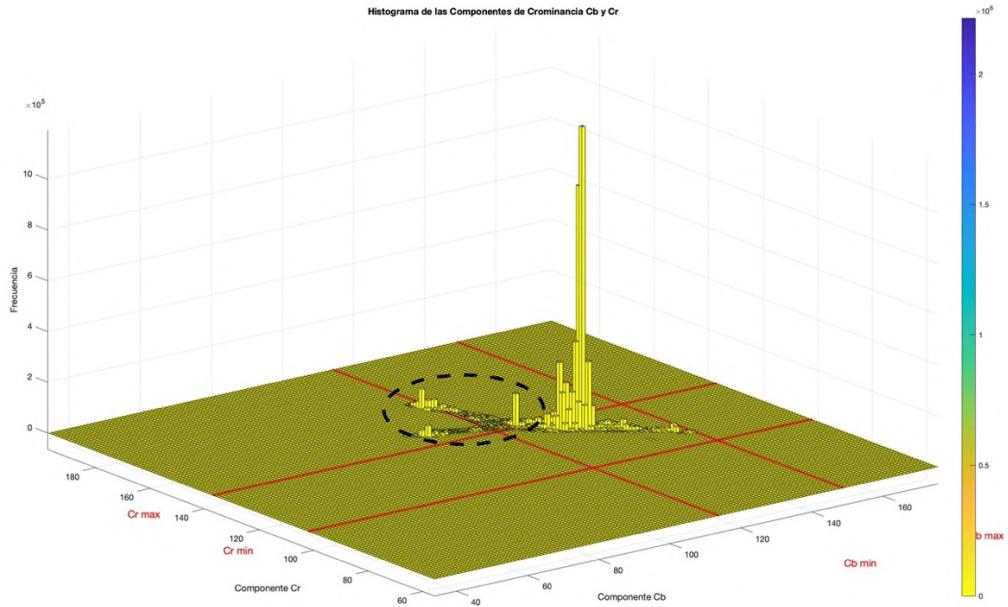
```

En un inicio un F-Score de 0.92025 nos pareció un resultado muy correcto, y las máscaras generadas eran, en su mayoría, muy claras donde lograba apreciarse con claridad la forma de todas ellas. No obstante, unas pocas máscaras eran totalmente incorrectas y no podrían tomarse como válidas a la hora de proseguir con la siguiente parte del proyecto.



Cómo puede apreciarse, pese a que se logra distinguir la forma en su mayoría todavía nuestro detector era algo impreciso, por lo que decidimos aplicar algunos cambios y mejoras al respecto. A continuación, se explicarán las mejoras realizadas y su justificación.

La mejora principal viene dada por la región de la que tomamos los supuestos píxeles de piel, esta región era la región rectangular óptima para lograr el mayor F-Score y por ello pudiera parecer que no hay forma de seleccionar una mejor región como nos ha mostrado nuestro propio optimizador, al menos no con ese tipo de delimitación. Y aun así si nos fijamos en el histograma, vemos que hay zonas de este donde se concentran cantidades de píxeles de piel no tan altas como las de nuestra región, pero bastante notables y probablemente significativas.



Es por ello por lo que la única solución que nos permite acotar esas regiones a la vez que acotamos nuestra región principal sin coger píxeles innecesarios es utilizar varias regiones y luego crear una máscara conjunta.

## 1. Múltiples regiones

Después de una búsqueda exhaustiva haciendo uso de nuestro algoritmo de optimización de umbrales hemos obtenido que las 5 regiones con mayor cantidad de píxeles pertenecientes a piel y por tanto las regiones que nos proporcionan un mayor F-Score son las siguientes:

```

Cb_min = 114; Cb_max = 149;
Cr_min = 102; Cr_max = 137;

Cb_min2 = 108; Cb_max2 = 115;
Cr_min2 = 155; Cr_max2 = 168;

Cb_min3 = 95;   Cb_max3 = 111;
Cr_min3 = 139;  Cr_max3 = 146;

Cb_min4 = 111;  Cb_max4 = 118;
Cr_min4 = 137;  Cr_max4 = 138;

Cb_min5 = 112;  Cb_max5 = 114;
Cr_min5 = 140;  Cr_max5 = 142;

```

Nuestro objetivo es lograr una máscara conjunta a partir de esas 4 regiones. Para ello primeros haremos breves cambios en la estructura del código para poderlo adaptar:

a. points matrix

Hasta ahora habíamos utilizado un vector de 4 posiciones con los 4 parámetros que delimitaban la región, sin embargo, ahora tenemos 5 conjuntos de 4 puntos, por lo que vamos a definir una matriz donde se almacenen los límites de todas las regiones, el procedimiento será el siguiente:

```
% Definir los vectores de puntos de control
points = [Cb_min, Cb_max, Cr_min, Cr_max];
points2 = [Cb_min2, Cb_max2, Cr_min2, Cr_max2];
points3 = [Cb_min3, Cb_max3, Cr_min3, Cr_max3];
points4 = [Cb_min4, Cb_max4, Cr_min4, Cr_max4];
points5 = [Cb_min5, Cb_max5, Cr_min5, Cr_max5];

% Definir la matriz points_matrix concatenando los
% vectores verticalmente
points_matrix = [points; points2; points3; points4; ...
    points5];
```

b. generateMasks(image, points\_matrix)

Para generar las máscaras de todas las regiones podemos definir una subfunción dentro de “algo3” que se encargue de generar máscaras de todas las regiones establecidas en la matriz y luego retornar las máscaras.

```
%-----
% Subfunción para generar máscaras
function masks = generateMasks(image, points_matrix)

    % Utilizar algo2 para generar máscaras para las
    % regiones
    masks = arrayfun(@(i) algo2(image, ...
        points_matrix(i, :)), 1:size(points_matrix, 1), ...
        'UniformOutput', false);

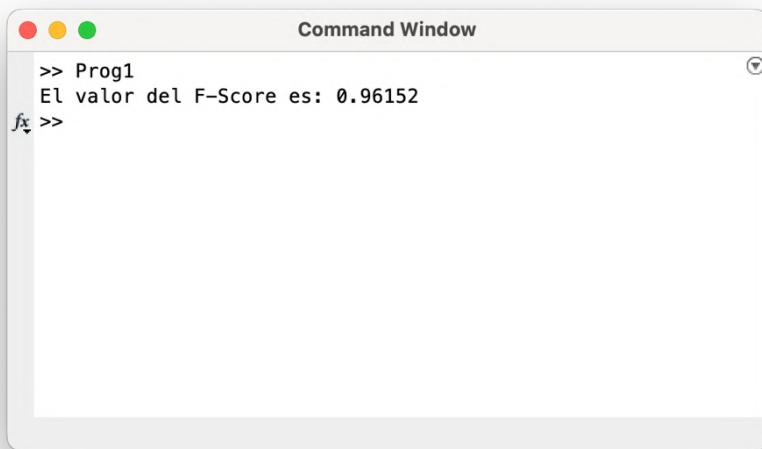
end
%-----
```

c. combineMasks(masks)

Y luego necesitaremos de una subfunción en “algo3” que combine todas las máscaras obtenidas una por región en una única máscara.

```
%-----%
% Subfunción para combinar máscaras
function combined_mask = combineMasks(masks)
    % Combinar las máscaras negando cualquier máscara
    % activa
    combined_mask = ~(any(cat(3, masks{:}), 3));
end
%-----%
```

Gracias al uso de múltiples regiones en vez de una única región para crear la máscara de piel, logramos obtener una eficiencia notablemente superior a la que habíamos logrado originalmente en el conjunto de entrenamiento. Aumentando el anterior 0.92025 a un mucho mayor 0.96152 valor de F-Score.



No obstante, pese a que ahora todas las máscaras tienen una clara forma de mano y un resultado mucho más parecido al ideal que el obtenido anteriormente, no quisimos conformarnos y decidimos aplicar más mejoras y que nos ayudaran a obtener mejores resultados.

A continuación, se relatan las demás estrategias y algoritmos que han resultado exitosos en el aumento de eficacia de nuestro sistema. Una vez explicadas todas ellas se expondrá el código completo de “algo3” donde se hallan aplicadas todas las nuevas mejores y adaptaciones.

## 2. Eliminar componentes pequeñas

Cuando analizamos detenidamente las máscaras binarias generadas podemos observar que un problema más o menos frecuente es la detección errónea de fragmentos que no pertenecen a la mano. A menudo estos fragmentos no están conectados a la mano sino en otra parte de la imagen, y algunos de ellos solo son pequeñas áreas que se han detectado como falso positivo en otros elementos de la imagen. Por ello podríamos analizar la máscara y encontrar todas las componentes de la máscara y quedarnos solo con la componente correspondiente a la mano, es decir, de todas las formas binarias que encontramos en la máscara quedarnos con la que corresponde a la mano y descartar las demás. Pues en el supuesto caso que tengamos áreas de falsos positivos estas áreas nunca serán más grandes que el área de la mano.

La mano corresponderá a la componente con el mayor número de píxeles ya que es extremadamente poco probable que haya un elemento detectado por el sistema del que su número de píxeles detectado como piel sea mayor que el número de píxeles detectados de la mano. Para que eso ocurriera nuestro detector debería ser extremadamente ineficiente, algo muy lejano a la realidad en este punto del desarrollo.

Para realizar esto desarrollamos una subfunción llamada `saveLargestConnectedComponent`. Esta subfunción tiene como objetivo guardar el componente más grande en una máscara binaria. La función toma dos argumentos: “`combined_mask`”, que es una máscara binaria que contiene múltiples componentes conectados, y “`output_filename`”, que es el nombre del archivo de salida donde se guardará el componente más grande. El proceso comienza encontrando todos los componentes en la máscara “`combined_mask`” utilizando la función “`bwconncomp`” y luego, calcula el número de píxeles en cada componente y encuentra el índice del componente más grande.

Después, crea una nueva máscara llamada “`largestComponent`” que contiene solo el componente más grande encontrado. Esta máscara se llena con “`true`” en los píxeles que pertenecen al componente más grande y “`false`” en los demás.

Finalmente, la función guarda la máscara “`largestComponent`” en el archivo especificado por “`output_filename`”. Si no se encuentra ningún componente en la máscara original, se guarda la máscara original en su lugar. La función devuelve un valor booleano “`sabe`” que indica si se pudo guardar el componente más grande o no.

```

%-----
% Subfunción para guardar el componente conectado más
% grande

function save = saveLargestConnectedComponent ( ...
    combined_mask, output_filename)

    % Encontrar componentes conectados en la máscara
    % combinada
    cc = bwconncomp(combined_mask);

    if cc.NumObjects > 0
        % Calcular el número de píxeles en cada componente
        numPixels = cellfun(@numel, cc.PixelIdxList);

        % Encontrar el índice del componente más grande
        [~, idx] = max(numPixels);

        % Crear una nueva máscara con solo el componente
        % más grande
        largestComponent = false(size(combined_mask));
        largestComponent(cc.PixelIdxList{idx}) = true;

        % Guardar la máscara en el directorio de salida
        imwrite(largestComponent, output_filename);

        save = true;

    else
        % No se encontraron componentes conectados,
        % guardar la máscara original
        imwrite(combined_mask, output_filename);
        save = false;
    end
end
%-----

```

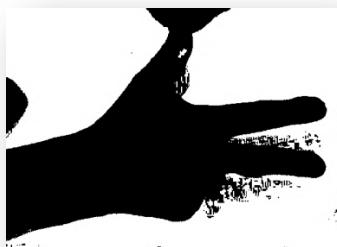
A pesar de que este procedimiento nos ayuda a limpiar en gran medida las máscaras, seguimos teniendo el problema de que gran parte de los elementos no pertenecientes a la mano, pero detectados igualmente están unidos a la mano, es decir, una zona que se detecte como piel justo al lado de la mano luego al convertir en máscara estará conectada a la máscara de la mano formando así una un área por lo que el procedimiento que acabamos de explicar no funcionaría en esos casos. A continuación de añade un ejemplo de lo recientemente comentado, donde una parte de cuello adyacente a la mano queda adherida a la misma área de la mano cuando creamos su máscara:



Máscara sin procesar



Máscara procesada



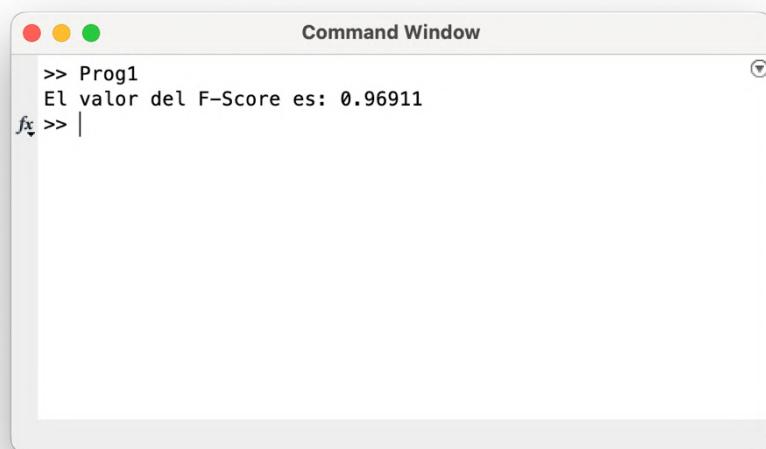
Máscara sin procesar



Máscara procesada

Cómo podemos observar, los pequeños puntos o pequeñas áreas que no están en contacto con la mano, y a las que podríamos llamar “ruido”, sí se eliminan con este procedimiento, además de otras notablemente grandes zonas de piel, no obstante, observamos que trozo de cuello del segundo ejemplo se mantiene igual por estar en contacto con la máscara principal.

Sin embargo, este proceso ha sido claramente exitoso mejorando la calidad de nuestras máscaras, algo que a su vez podemos cuantificar si nos fijamos en el valor de F-Score del conjunto de entrenamiento donde hemos aumentado el F-Score de 0.96152 a 0.96911.



### 3. Apertura y umbral de apertura

Como bien hemos visto, cuando tratamos de aislar la máscara de la mano, a menudo se detectan píxeles de otros elementos no deseados junto con los de la propia mano. Los esfuerzos aplicados hasta este punto nos permiten separar los elementos que no estén conectados al área principal de la máscara donde esté la mano, pero no a los que estén en contacto por algún píxel. Es por eso por lo que debemos abordar este problema con una nueva perspectiva que a su vez pueda complementarse con el punto anterior. Por ello, nos propusimos utilizar dos operaciones morfológicas: erosión y dilatación.

La erosión es una operación morfológica que, en esencia, "encoge" los objetos blancos en una imagen y expande los negros. Lo que hace es eliminar píxeles en el borde de los objetos, lo que puede separar mejor los píxeles de la mano de los de otros elementos con el procedimiento anterior. Y la erosión sería el proceso inverso, encoge los negros y expande los blancos.

Es importante destacar que las máscaras que nosotros creamos están invertidas, es decir, nosotros con algo2 creamos máscaras donde la zona blanca corresponde a la mano y la zona negra al fondo, y, posteriormente, las invertimos al final del algoritmo algo3 para poder evaluar su eficiencia. Pero estos algoritmos que estamos explicando se aplican sobre imágenes del siguiente estilo:



Es por ello por lo que aplicamos erosión y luego dilatación (apertura morfológica) y no al revés (cierre morfológico).

Volviendo al algoritmo en sí mismo, la idea sería aplicar lo que se llama un proceso de apertura, es decir, aplicar una erosión y luego una dilatación para recuperar la forma. Pero aplicando entre medio de ambas operaciones la operación de "tomar el área más grande" que hemos realizado antes para limpiar las máscaras. La idea detrás de esta operación es conservar solo la parte más grande de la máscara, que generalmente es

la de la mano, eliminando así elementos no deseados. La erosión nos ayudará a separar mejor los elementos antes de quedarnos con el más grande.

No obstante, la apertura no reconstruye la imagen perfectamente, el proceso tanto de erosión como dilatación tienden a eliminar detalles y puede distorsionar la forma de la máscara de la mano. Pero debíamos comprobar si la eliminación de elementos interferentes compensaba la distorsión que nos causaba este proceso.

Sin embargo, descubrimos que esta estrategia no siempre funciona bien. En imágenes donde las máscaras ya eran casi ideales, sin elementos no deseados, la erosión y la dilatación empeoran la forma de la máscara. Y las pocas imágenes donde si se eliminaban elementos interferentes no compensaban la distorsión que nos aportaba el proceso. Esto nos llevó a un F-Score ligeramente más bajo en vez de mayor.

Por otro lado, observamos como en imágenes con más elementos no deseados, esta estrategia ayudó. Pues, aunque la forma de la mano se veía perjudicada, la eliminación de elementos compensaba y su F-Score resultaba ser mayor. Pero necesitábamos una forma de diferenciar entre las imágenes que se beneficiarían de este proceso y las que no.

Entonces, investigamos más y encontramos un criterio útil: el tamaño de la máscara. Observamos que en imágenes donde había muchos elementos no deseados junto a la mano, el tamaño de la máscara era notablemente mayor, la mayoría superando los 100,000 píxeles. Un valor que podemos utilizar como punto de corte, aunque posteriormente definiremos un algoritmo para afinarlo.

La estrategia final fue aplicar erosión, "tomar el área más grande" y luego dilatar las máscaras con más de un cierto número de píxeles, y en el resto de las imágenes solo "tomar el área más grande" sin realizar la apertura.

Esta estrategia nos ayudó a limpiar las máscaras en imágenes donde había muchos elementos no deseados sin afectar negativamente las máscaras que ya eran casi ideales. Obteniendo una clara mejora del F-Score.

El código en cuestión es el siguiente:

```
% Calcular el número de píxeles en cada componente
numPixels = cellfun(@numel, cc.PixelIdxList);

% Comprobar si el componente mayor es demasiado grande
%*Indicio de píxeles de piel no pertenecientes a la mano*
if max(numPixels) > umbralApertura
    % Si es así, realiza una erosión para reducir
    % el tamaño del componente
```

```

se = strel('sphere', 10);
    % Esfera de radio 10

combined_mask = imerode(combined_mask, se);

%Encontrar las nuevas componentes
cc = bwconncomp(combined_mask);
numPixels = cellfun(@numel, cc.PixelIdxList);
apertura = true;
end

% Encontrar el índice del componente más grande
[~, idx] = max(numPixels);

% Crear una nueva máscara con solo el componente
%más grande
largestComponent = false(size(combined_mask));
largestComponent(cc.PixelIdxList{idx}) = true;

% Si se realizó una erosión previamente, realiza
una dilatación para restaurar el tamaño
if apertura
    largestComponent = imdilate(largestComponent, se);
end

```

En el código, notamos la declaración `strel('sphere', 10)`, la cual hace referencia al elemento estructural utilizado en el proceso de apertura. En este caso, se trata de un elemento esférico con un radio de 10 píxeles. Este elemento es fundamental, ya que se emplea tanto en la operación de erosión como en la de dilatación. Su función principal consiste en determinar la extensión espacial alrededor de un píxel en la máscara que se tendrá en cuenta durante las operaciones morfológicas de erosión y dilatación.

La elección de este elemento en concreto, al igual que la elección de su radio, no han sido arbitrarias, sino que se ha determinado empíricamente después de un proceso de pruebas exhaustivas. Este elemento concreto ha demostrado ser el más eficaz para este proceso al igual que el valor de su radio específico ha demostrado producir resultados óptimos en términos de F-Score, superando tanto a valores de radio más pequeños (como 9) como a valores más grandes (como 11) como a otros elementos estructurantes.

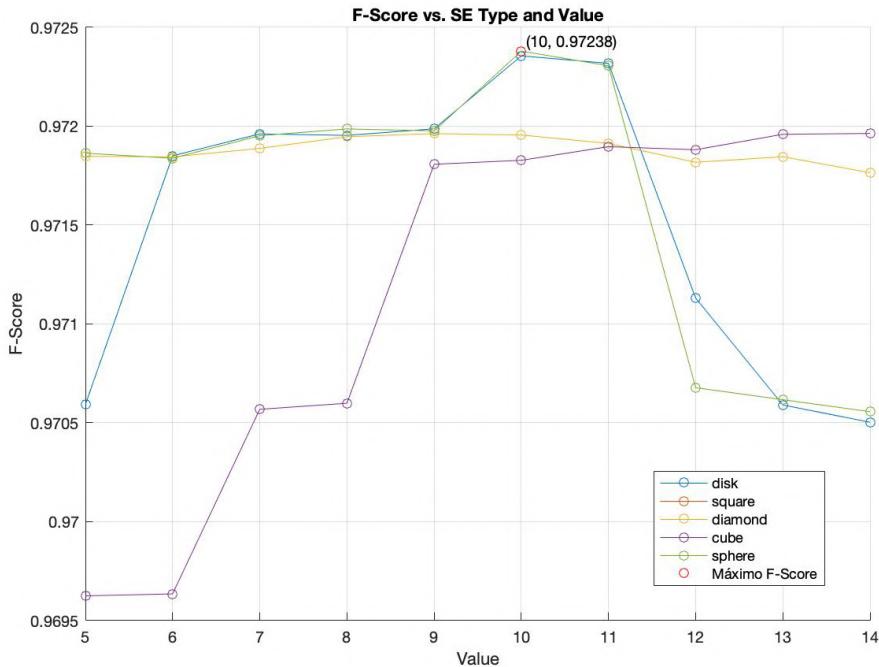
El código correspondiente al testeo de elementos estructurantes es el siguiente:

```

1 % Tipos de elementos estructurantes y valores
2 se_types = {'disk', 'square', 'diamond', 'cube', 'sphere'};
3 values = 0:7;
4
5 % Matrices para almacenar los resultados
6 f_scores = zeros(length(se_types), length(values));
7
8 for i = 1:length(se_types)
9     for j = 1:length(values)
10        se = se_types{i};
11        value = values(j);
12
13        algo3SE(points_matrix, dataset_type, umbralApertura, processingOptions, ...
14                 se, value);
15
16        % Llama a la función algo4 y captura el F-Score
17        FScore = algo4Graphics(dataset_type, false);
18
19        % Almacena el F-Score en la matriz
20        f_scores(i, j) = FScore;
21    end
22 end
23
24 % Encuentra el valor máximo de F-Score y su ubicación
25 [max_f_score, max_index] = max(f_scores(:));
26 [se_index, value_index] = ind2sub(size(f_scores), max_index);
27 optimal_se = se_types{se_index};
28 optimal_value = values(value_index);
29
30 % Gráfica
31 figure;
32 hold on;
33 for i = 1:length(se_types)
34     plot(values, f_scores(i, :), 'o-', 'DisplayName', se_types{i});
35 end
36
37 % Marcar el punto máximo
38 plot(optimal_value, max_f_score, 'ro', 'DisplayName', 'Máximo F-Score');
39
40 hold off;
41
42 xlabel('Value');
43 ylabel('F-Score');
44 legend('Location', 'Best');
45 title('F-Score vs. SE Type and Value');
46 grid on;
47
48 % Mostrar el valor máximo en la gráfica
49 text(optimal_value, max_f_score, ...
50      [' (' num2str(optimal_value) ', ' num2str(max_f_score) ')'], ...
51      'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'left');
```

Este código nos permite comprobar empíricamente que elementos y valores son los más adecuados, evaluando el procedimiento de umbral de apertura para diferentes elementos estructurantes y diferentes valores.

La gráfica resultante fue la siguiente:

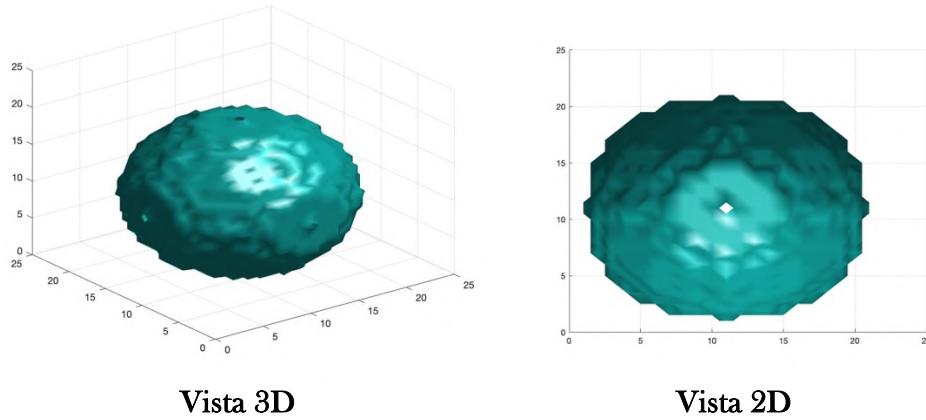


Las pruebas realizadas nos muestran como el mejor elemento estructurante para nuestro propósito es una esfera ligeramente por encima del círculo.

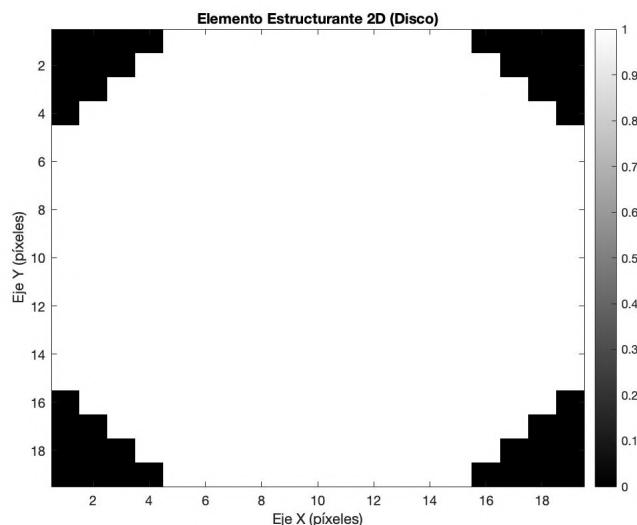
Es importante destacar que el elemento estructural 'sphere' (esfera) es un elemento en tres dimensiones (3D), mientras que nuestras imágenes son bidimensionales (2D). A primera vista, podría parecer incompatible aplicar un elemento 3D a datos 2D. Sin embargo, MATLAB realiza una conversión implícita para permitir su aplicación en un espacio 2D. Esta conversión da como resultado un elemento estructural que es similar a un kernel circular, pero con una forma ligeramente distinta.

Al convertirse en 2D, la 'sphere' se asemeja a un círculo, específicamente al que obtendríamos con la función `strel('disk', 10)`. Es por eso, que los resultados obtenidos al aplicar este elemento 3D convertido, en nuestro proceso de apertura, son muy parecidos a los obtenidos con el elemento circular típico, sin embargo, ligeramente superiores en términos de rendimiento, particularmente en la métrica de F-Score, como nos muestran pruebas realizadas. Por lo tanto, se eligió utilizar el elemento 'sphere' en lugar del círculo, ya que, al convertirse en 2D, su forma ligeramente distinta proporciona resultados más efectivos.

**Elemento Estructurante 3D (Esfera)  
con radio de 10 píxeles**



**Elemento Estructurante 2D (Kernel circular)  
con radio de 10 píxeles**



Asimismo, el hecho de que el elemento estructural tenga una forma circular es algo lógico, pues esto nos permite que se ajuste de manera más natural a la forma de la mano, que tiende a ser una estructura aproximadamente redondeada. Lo que significa que el elemento esférico en 2 dimensiones o el propio círculo, son los más apropiado para capturar y preservar la estructura esencial de la mano mientras se eliminan elementos no deseados. Si se utilizara un kernel rectangular, por ejemplo, se podrían perder detalles importantes en la forma de la mano, ya que un

kernel rectangular tendería a cortar esquinas y bordes en lugar de mantener la forma general redondeada de la mano.

Para terminar, debemos también buscar un umbral de apertura que sea más preciso, sabiendo que este estaría alrededor de los 100,000 píxeles. Para ello creamos un algoritmo llamado “testApertura.m” que nos proporcionaría el valor óptimo:

```

1
2 Cb_min = 114; Cb_max = 149;
3 Cr_min = 102; Cr_max = 137;
4
5 Cb_min2 = 108; Cb_max2 = 115;
6 Cr_min2 = 155; Cr_max2 = 168;
7
8 Cb_min3 = 95; Cb_max3 = 111;
9 Cr_min3 = 139; Cr_max3 = 146;
10
11 Cb_min4 = 111; Cb_max4 = 118;
12 Cr_min4 = 137; Cr_max4 = 138;
13
14 Cb_min5 = 112; Cb_max5 = 114;
15 Cr_min5 = 140; Cr_max5 = 142;
16
17 dataset_type = 'Training';
18
19 % Definir los vectores de puntos de control
20 points = [Cb_min, Cb_max, Cr_min, Cr_max];
21 points2 = [Cb_min2, Cb_max2, Cr_min2, Cr_max2];
22 points3 = [Cb_min3, Cb_max3, Cr_min3, Cr_max3];
23 points4 = [Cb_min4, Cb_max4, Cr_min4, Cr_max4];
24 points5 = [Cb_min5, Cb_max5, Cr_min5, Cr_max5];
25
26 % Definir la matriz points_matrix concatenando los vectores verticalmente
27 points_matrix = [points; points2; points3; points4; points5];
28
29 % Inicializa variables para almacenar el mejor F-Score y el umbral correspondiente
30 bestFScore = 0;
31 bestThreshold = 0;
32
33 UmbralSup = 90000;
34 UmbralInf = 120000;
35 Paso = 1;
36
37 % Define un rango de umbrales que deseas probar
38 thresholds = UmbralInf:Paso:UmbralSup;
39
40 % Calcular el número total de iteraciones
41 total_iterations = (UmbralSup-UmbralInf)/Paso;
42
43 % Inicializar el contador de iteraciones completadas
44 completed_iterations = 0;
45
46 % Crear la barra de progreso
47 h = waitbar(0, 'Procesando...');
48
49 for threshold = thresholds
50     % Llama a la función algo3 con el valor actual de threshold
51     algo3(points_matrix, dataset_type, threshold, false);
52
53     % Llama a la función algo4 con dataset_type
54     FScore = algo4(dataset_type);
55
56     % Verifica si el F-Score actual es mejor que el mejor F-Score
57     % encontrado hasta ahora
58     if FSFscore > bestFScore
59         bestFScore = FSFscore;
60         bestThreshold = threshold;
61     end
62     % Incrementar el contador de iteraciones completadas
63     completed_iterations = completed_iterations + 1;
64
65     % Calcular el progreso en porcentaje
66     progress_percent = (completed_iterations / total_iterations) * 100;
67

```

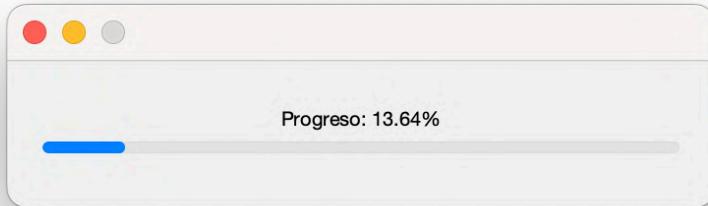
```

68      % Actualizar la barra de progreso
69      waitbar(progress_percent / 100, h, sprintf('Progreso: %.2f%%', progress_percent));
70
71 end
72
73 % Cerrar la barra de progreso
74 close(h);
75
76 % Imprime el mejor F-Score y el umbral correspondiente
77 disp(['El mejor valor del F-Score es: ' num2str(bestFScore)]);
78 disp(['El umbral correspondiente es: ' num2str(bestThreshold)]);

```

Este algoritmo tiene como objetivo determinar el umbral a partir del cual la operación de apertura comentada en este apartado es útil para mejorar las máscaras.

El algoritmo dispone de una barra de progreso que facilita al usuario visualizar en porcentaje el tiempo de ejecución del algoritmo.



En cuanto al propio algoritmo, primeramente, se definen varios conjuntos de puntos de control, cada uno especificando los rangos `Cb_min`, `Cb_max`, `Cr_min` y `Cr_max` que definen la piel en el espacio de color YCbCr. Se crea una matriz llamada “`points_matrix`” que concatena estos conjuntos de puntos de control como hemos visto antes.

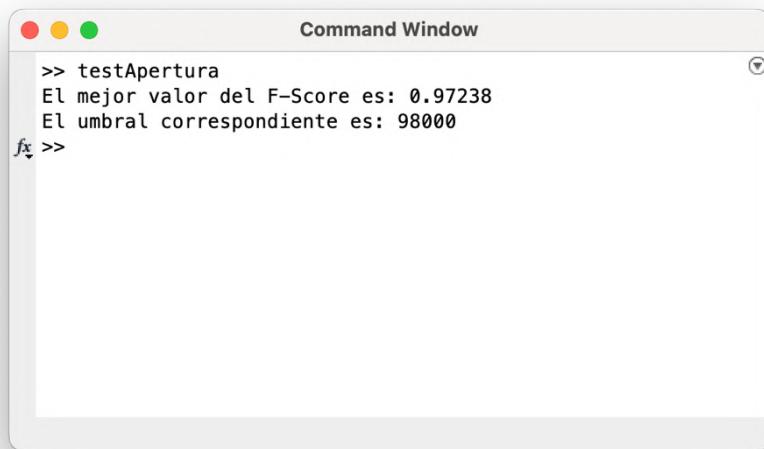
Se establecen los valores de `UmbralSup`, `UmbralInf` y `Paso`, que definen un rango de umbrales a probar. El código itera a través de los umbrales en el rango definido y realiza los siguientes pasos:

- Llama a la función “`algo3`” con los puntos de control y el umbral actual.
- Llama a la función “`algo4`” con el tipo de conjunto de datos.

Se compara el F-Score calculado con el mejor F-Score encontrado hasta el momento. Si el F-Score actual es mejor, se actualiza la variable “`bestFScore`” y se registra el umbral correspondiente en “`bestThreshold`”.

Luego se muestra el mejor F-Score obtenido y el umbral correspondiente.

Gracias a este sistema hemos logrado obtener que el mejor umbral para utilizar este sistema de eliminación de elementos interferentes usando la apertura como operación auxiliar se encuentra en los 98,000 píxeles. Y aumentando el valor de F-Score del conjunto de entrenamiento de 0.96152 a 0.97238.



The image shows a MATLAB Command Window titled "Command Window". The window contains the following text:  
>> testApertura  
El mejor valor del F-Score es: 0.97238  
El umbral correspondiente es: 98000  
fx >>

Además, si observamos las máscaras resultantes podemos observar una clara limpieza en zonas con mucha presencia de píxeles interferentes:



Máscara sin procesar



Máscara procesada

#### 4. Relleno de regiones

Después de separar las áreas de las manos de otros elementos no deseados, podemos observar que algunas de estas áreas de las manos contenían espacios vacíos o "huecos" que resultaban en falsos negativos. Aunque era un problema que no afectaba a la forma o al contorno, decidimos igualmente abordar este problema para así mejorar la métrica F-Score y retocar nuestras máscaras.

El código siguiente se encarga de llenar los huecos en la máscara de la mano:

```
% Rellenar los huecos en la máscara  
filledMask = imfill(largestComponent, 'holes');
```

Esta operación, “imfill”, se utiliza para completar o llenar las áreas vacías dentro de la región que nosotros previamente ya hemos identificado como la mano, lo que contribuye a reducir falsos negativos y mejorar la precisión general de la máscara. A continuación, se añade un ejemplo donde se aprecia claramente la utilidad de esta técnica:

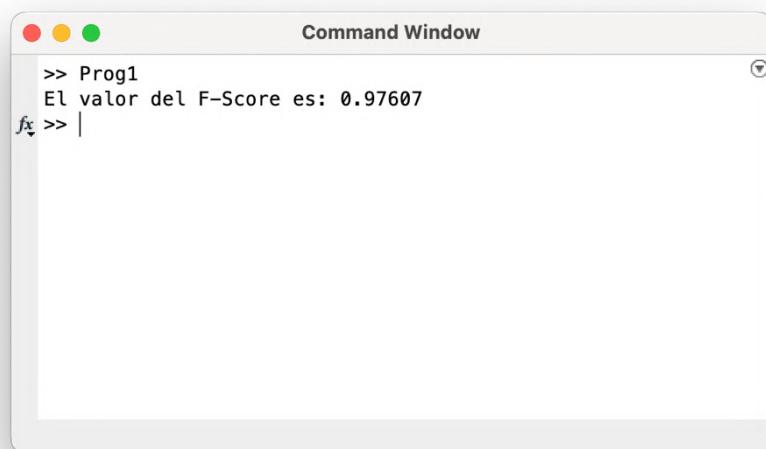


Máscara sin procesar



Máscara procesada

Esta mejor nos permitió aumentar el valor de F-Score del conjunto de entrenamiento de 0.97238 a 0.97607.

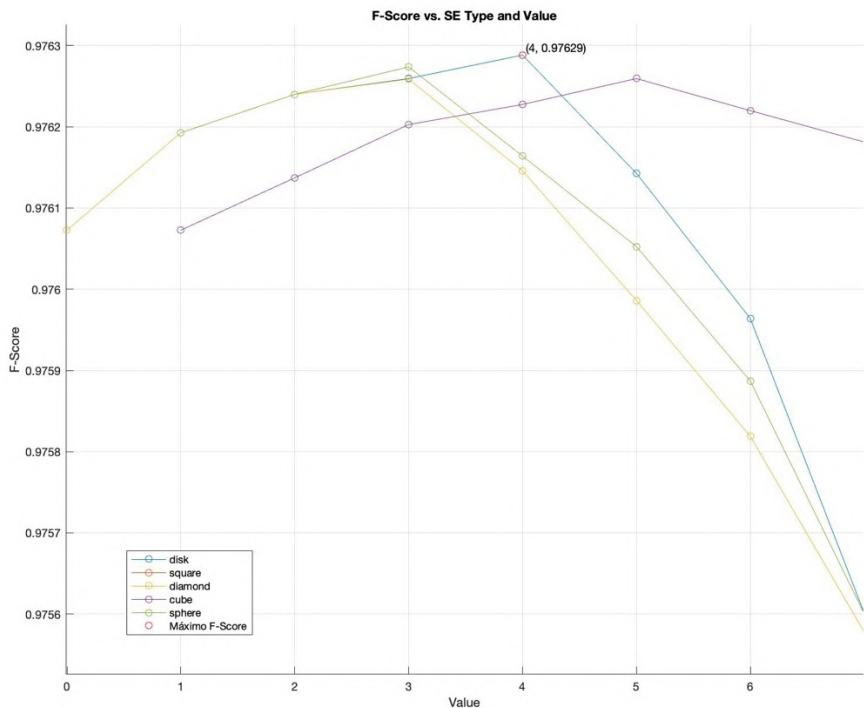


## 5. Suavizado por apertura

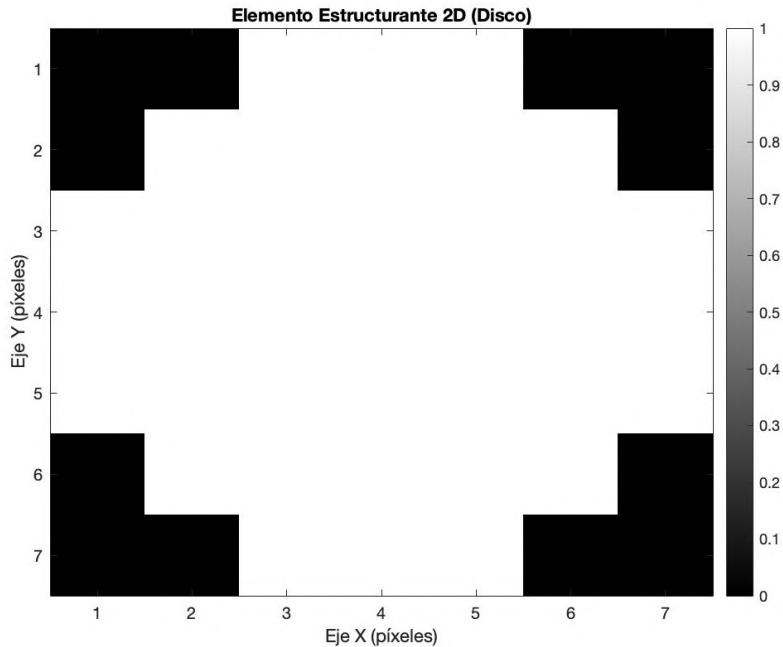
Luego, se nos ocurrió que quizás podríamos mejorar la apariencia de la máscara de manos al suavizar sus bordes, lo que la haría más parecida a la máscara ideal. Entonces recordamos el proceso que habíamos utilizado anteriormente, la "apertura". La apertura consiste en realizar primero una erosión y luego una dilatación en una imagen, lo que generalmente conduce a la pérdida de algunos detalles y una leve alteración de la forma, pero al mismo tiempo suaviza los elementos.

Por lo tanto, se nos ocurrió la idea de aplicar una apertura, pero esta vez con un elemento estructurante de menor tamaño. La intención era lograr un suavizado ligero que afectara mínimamente la imagen, pero que redujera las protuberancias en los bordes. La elección del tamaño y del tipo de elemento se basó en el algoritmo utilizado anteriormente para determinar el mejor elemento estructurante y el mejor valor de este en el procedimiento de eliminación de elementos interferentes por apertura. Pero esta vez para determinar cuál generaba un suavizado más coherente con la imagen original. Estas pruebas evaluaron el impacto de las diferentes opciones en el F-Score, y se seleccionó el elemento y valor que proporcionó el mejor resultado.

Los resultados fueron los siguientes:



El resultado óptimo, en este caso, se obtiene utilizando un elemento estructurante de tipo disco (`strel('disk')`) con un radio de 4 píxeles. Es decir, un Kernel circular de radio 4. Este elemento nos proporciona la mejor métrica F-Score y nos permite suavizar los bordes de la máscara, mejorando su apariencia sin comprometer significativamente la forma de los objetos de interés.



A continuación, la subfunción que suaviza las máscaras:

```
%-----%
% Subfunción para suavizar los bordes por apertura
function smoothed_mask = smoothEdges(mask)
    % Crear un elemento estructurante (kernel) para
    % suavizado de bordes
    se = strel('disk', 4);
    % Puede ajustarse el tamaño del kernel

    % Aplicar dilatación seguida de erosión para
    % suavizar los bordes
    smoothed_mask = imerode(imdilate(mask, se), se);
end
%-----%
```

La operación de apertura mejoró la apariencia de los bordes en las máscaras de manos sin afectar drásticamente a su forma general:



Máscara sin procesar



Máscara procesada

Cómo se puede observar la mejora no es muy notable pero sí se aprecia como el contorno ha quedado mejor definido, ahora las partes muy afiladas de los bordes han sido suavizadas.

Esta mejor no aumentó significativamente el valor de F-Score, en el caso del conjunto de entrenamiento de 0.97607 a 0.97629, pero sí logró mejorar ligeramente el contorno en la mayoría de los casos manteniendo la forma original.

A screenshot of a MATLAB Command Window titled "Command Window". The window shows the following text:  
>> Prog1  
El valor del F-Score es: 0.97629  
fx >> |

## 6. Reducción de sombras

Una vez todas estas mejoras fueron aplicadas logramos obtener un F-Score del 0.95 en el conjunto de entrenamiento, no obstante, no se trataba solo de obtener una buena métrica en ese parámetro, sino que el objetivo base de este sistema era crear máscaras que luego nos permitieran hacer un conteo de dedos. Al observar las máscaras vimos que prácticamente todas ellas permitían, al menos a simple vista, contar con precisión el número de dedos de la mano. Sin embargo, observamos un último problema, las sombras.

El detector a veces falla y detecta algunas sombras de manos como partes de la mano debido a las limitaciones en el procesamiento de imágenes y el uso de un modelo basado en el espacio de color YCbCr. El espacio de color YCbCr se basa en componentes de luminancia (Y) y crominancia (Cb y Cr) y, cuando una sombra de mano cae sobre otro elemento, puede cambiar la luminancia de esa región, lo que, dependiendo del color de ese elemento, podría hacer que el detector la interprete como piel. Dando lugar, en algunos casos, a componentes crominantes (Cb y Cr) que son similares a los de la piel.

Para ejemplificar este hecho veamos la siguiente imagen:



La imagen usada como ejemplo tiene un fondo con una crominancia muy similar a la de la piel, no obstante, como las regiones de detección han estado bien delimitadas nuestro sistema es capaz de distinguir los píxeles de la mano de la camiseta de fondo. No obstante, las zonas donde la sombra cae sobre la camiseta parecen tener unas componentes Cb y Cr extremadamente parecidos a las de la de los píxeles de piel y nuestro sistema los detecta como parte de la mano resultando en máscaras de manos con más dedos que los de la imagen original. Para analizar este suceso vamos a separar la imagen anterior en los tres canales del espacio de color YCbCr y observar los resultados:

**Imagen 1: Canal Y (Luminancia)**



Si nos fijamos en el canal Y, donde se representa la luminancia, vemos que sí podemos observar a simple vista la diferencia entre sombras y piel, no obstante, nuestro sistema se basa en la detección de píxeles en función del color y esta componente no nos aporta ninguna información del color de la mano. Un sistema riguroso debería ser capaz de detectar de forma precisa todos los píxeles de mano independiente de la información de la luminancia.

**Imagen 2: Canal Cb (Crominancia Azul)**



En el canal Cb, que, en la mayoría de los casos, permite una fácil distinción de los píxeles, surge un escenario peculiar. En este caso, tanto las zonas de sombra como las áreas de piel exhiben valores crominantes extremadamente similares. Aunque el fondo logra mantener su distinción del resto de la imagen de forma sutil, la situación cambia con los píxeles de sombra. Estos píxeles, en lugar de destacarse claramente, se entremezclan con las regiones de la mano en términos de crominancia azul.

**Imagen 3: Canal Cr (Crominancia Roja)**



El canal Cr, que se encarga de representar las diferencias en la crominancia roja, arroja luz sobre el problema subyacente. Los píxeles correspondientes a la camiseta se distinguen considerablemente de los píxeles de la mano en términos de crominancia. No obstante, los píxeles en áreas de sombras y de piel exhiben, de nuevo, una sorprendente similitud, dejando en evidencia la razón por la que nuestro detector no es capaz de realizar una distinción clara

Abordar la eliminación de los píxeles de sombra en nuestras máscaras se convierte en el último y crítico desafío que debemos superar. Si el modelo de detección no toma en consideración estas variaciones, existe el riesgo de que clasifique erróneamente las sombras como piel, lo que tendría un impacto perjudicial en nuestro sistema de detección. A menudo las optimizaciones realizadas en modelos de detección de piel no son lo suficientemente sofisticadas para diferenciar de manera precisa las sombras de la piel, principalmente debido a su limitación para adaptarse a cambios en la iluminación. Pero es nuestra responsabilidad afinar el sistema para que sea capaz de distinguir este tipo de píxeles correspondientes a las sombras a partir de la base de datos disponible.

La estrategia que implementamos fue la siguiente: a partir de las imágenes donde detectamos errores en la detección debidos a sombras, creamos un nuevo banco de datos con únicamente con recortes de las áreas correspondientes a los píxeles de sombra detectado como piel en las imágenes. Luego, generamos un histograma de estos píxeles, siguiendo el mismo proceso utilizado en algo1 para los píxeles de piel.

A continuación, similar a lo que hicimos para optimizar el umbral de detección de los píxeles de piel, llevamos a cabo una búsqueda exhaustiva

para identificar las regiones más pequeñas que contenían la mayor concentración de píxeles de sombra, pero que incluyeran la menor cantidad posible de píxeles de piel. Esto nos permitió obtener pequeñas regiones que se encontraban dentro de las áreas que habíamos definido como piel, pero que en esas regiones específicas predominaban los tonos típicos de sombra en lugar de los de piel.

Dentro de la función "algo3", diseñamos una matriz llamada "points\_removal\_matrix" que alberga las regiones más comunes con tonos de sombras. De esta manera, si el usuario opta por habilitar la "eliminación de sombras", puede suprimir estas áreas de las máscaras resultantes.

```
% Definimos una matriz con las regiones donde se
% concentran las sombras
if shadowsRemoval
    % Agregar una matriz de conjuntos para
    % eliminación
    points_removal_matrix = [[122, 122, 139, 141]; ...
                             [120,120,142,142]; ...
                             [121,121,140,140]; ...
                             [113,117,139,139]; ...
                             [123,123,139,139]];

    % Procesar imágenes con eliminación de sombras
    processImages(points_matrix, ...
                  points_removal_matrix, input_dir, ...
                  output_dir, umbralApertura, ...
                  shadowsRemoval);
else
    % Procesar imágenes sin eliminación de sombras
    processImages(points_matrix, [], input_dir, ...
                  output_dir, umbralApertura, ...
                  shadowsRemoval);
end
```

Luego, utilizando el mismo enfoque que usamos para crear la máscara de piel, generamos una máscara conjunta de sombras a partir de todas las máscaras de sombra. Posteriormente, sustraemos esta máscara conjunta de la máscara de piel, eliminando así los falsos positivos.

```
if shadowsRemoval
    % Generar máscaras para todas las regiones
    masks_removal = generateMasks(image, ...
                                    points_removal_matrix);

    % Combinar las máscaras
    combined_removal_mask = ...
        combineMasks(masks_removal);
```

```
% Restarla
combined_mask = combined_mask & ...
    combined_removal_mask;
end
```

El resultado de esta estrategia son imágenes mucho mejor definidas con una menor cantidad de falsos positivos y una métrica F-Score mejorada, que aumentó, el caso del conjunto de entrenamiento, de 0.97629 a 0.9804.

No obstante, la mejor forma de observar los beneficios de esta mejora es fijándonos en las máscaras obtenidas, para ellos usaremos la imagen que en un inicio hemos usado como ejemplo de este desafío:



**Imagen original**



**Máscara sin reducción  
de sombras**



**Máscara con reducción  
de sombras**

O en otros ejemplos como el siguiente donde también vemos una diferencia significativa:



Imagen original

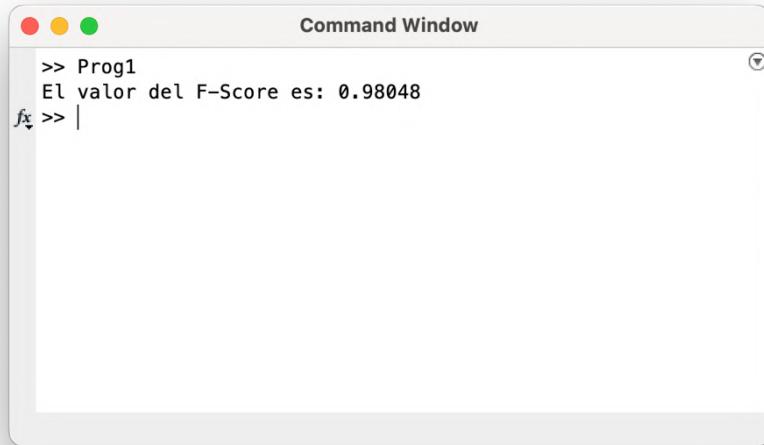


Máscara sin reducción  
de sombras



Máscara con reducción  
de sombras

Es evidente que la reducción de sombras ha tenido un impacto significativo en la mejora de las máscaras, acercándolas aún más a la forma de la imagen original. Hemos eliminado dedos ficticios causados por falsos positivos y hemos logrado un contorno más preciso. Aunque persisten algunos fragmentos que se han mantenido debido a la similitud cromática con la mano, la diferencia entre las imágenes es notablemente mejor y la métrica de F-Score lo corrobora.



Existe la posibilidad, que en bancos de imágenes donde no existan sombras que se confundan con la mano, este procedimiento será innecesario e incluso podría ser perjudicial de manera sutil. Por lo tanto, ofrecemos al usuario la opción de activar o desactivar la reducción de sombras durante la ejecución del sistema según crea conveniente.

Luego de implementar estas mejoras, hemos transformado significativamente nuestro algoritmo "algo3" inicial en una versión considerablemente más eficaz. Este nuevo algoritmo incorpora el uso de múltiples regiones para la detección de píxeles de piel, la eliminación de pequeños componentes interferentes, el relleno de regiones con falsos negativos, operaciones morfológicas destinadas a mejorar la eliminación de elementos no deseados en máscaras con mayor número de posibles falsos positivos, un proceso de apertura para suavizar la forma, y un sistema de reducción de sombras. Estas mejoras han sido implementadas con el objetivo de aumentar la eficiencia del sistema y producir máscaras óptimas para la siguiente etapa del proceso.

Además, se ha añadido la funcionalidad de poder o no activar ciertas operaciones de postprocesamiento a gusto del usuario, por si se diera el caso que las considera innecesarias o perjudiciales. Cabe decir que después de nuestro propio análisis sugerimos activarlas todas ya que nuestras pruebas han demostrado su eficacia en la gran mayoría de casos. No obstante, un mayor banco de datos sería necesario para verificar la veracidad de esta afirmación en todos los casos posibles.

A continuación, se proporciona el código correspondiente a la versión mejorada de "algo3":

```

function algo3(points_matrix, dataset_type, umbralApertura,
processingOptions)

1 function algo3(points_matrix, dataset_type, umbralApertura, processingOptions)
2 %
3 %
4 % algo3 - Genera máscaras de detección de piel para un conjunto de imágenes.
5 %
6 %
7 %
8 % Entradas:
9 % - points_matrix: Matriz de 4 puntos de control (Cb_min, Cb_max, Cr_min,
10 % Cr_max) para cada región.
11 %
12 % - dataset_type: Directorio del conjunto de datos('Training' o 'Validation').
13 %
14 % - umbralApertura: Umbral para el proceso de apertura (ajuste personalizado).
15 %
16 % - processingOptions: Vector de booleanos que controlan varias opciones de
17 % procesamiento.
18 %
19 % - processingOptions(1): Habilita o deshabilita la reducción de sombras.
20 % - processingOptions(2): Habilita o deshabilita el relleno de la máscara.
21 % - processingOptions(3): Habilita o deshabilita el suavizado de bordes de
22 % la máscara.
23 %
24 %
25 %
26 % Explicación de la función:
27 % La función algo3 utiliza la función algo2 para generar máscaras de detección
28 % de piel para un conjunto de imágenes. El conjunto de imágenes se define
29 % mediante un directorio ("Images") dentro del directorio del conjunto de datos
30 % especificado. Las máscaras se generan y se escriben en un directorio llamado
31 % "Masks" con el mismo nombre que las imágenes procesadas. El umbral de apertura
32 % se utiliza para controlar el proceso de apertura morfológica y ajustar la
33 % detección de piel. Los elementos de processingOptions controlan varias opciones
34 % de procesamiento.

```

```

35 %
36 %
37 %
38 % Ejemplo de uso:
39 % points_matrix = [120, 140, 120, 140;      % Puntos de control de la región 1
40 %                   100, 120, 150, 170;      % Puntos de control de la región 2
41 %                   110, 130, 160, 180;      % Puntos de control de la región 3
42 %                   ...
43 %                   90, 110, 130, 150];     % Puntos de control de la región N
44 %
45 %
46 % dataset_type = 'Training'; % Especifica el directorio del conjunto de datos
47 % umbralApertura = 98000;    % Ajusta el umbral de apertura
48 % processingOptions = [true, true, true]; % Habilita todas las opciones de
49 %                               procesamiento
50 %
51 % Llama a la función para generar máscaras.
52 % algo3(points_matrix, dataset_type, umbralApertura, processingOptions);
53 %
54 %
55 %
56 % Verificar los argumentos de entrada
57 dataset_type = checkInputArguments(points_matrix, dataset_type);
58 %
59 % Configurar directorios
60 [input_dir, output_dir] = setupDirectories(dataset_type);
61 %
62 %
63 % Creamos o no la matriz de eliminación de sombras
64 if processingOptions(1)
65     % Agregar un matriz de conjuntos para eliminación
66     points_removal_matrix = [[122, 122, 139, 141]; ...
67                             [120, 120, 142, 142]; ...
68                             [121, 121, 140, 140]; ...
69                             [113, 117, 139, 139]; ...
70                             [123, 123, 139, 139]];
71 %
72 % Procesar imágenes con eliminación de sombras
73 processImages(points_matrix, points_removal_matrix, input_dir, ...
74               output_dir, umbralApertura, processingOptions);
75 %
76 else
77     % Procesar imágenes sin eliminación de sombras
78     processImages(points_matrix, [], input_dir, ...
79                   output_dir, umbralApertura, processingOptions);
80 end
81 %
82 end
83 %
84 %
85 %
86 %-----SUBFUNCIONES-----
87 %
88 % Subfunción para verificar los argumentos de entrada
89 function dataset_type = checkInputArguments(points_matrix, dataset_type)
90     % Comprobar si se proporcionó la matriz de puntos
91     if nargin < 1
92         error(['Debes proporcionar una matriz de puntos de control ' ...
93               '[Cb_min, Cb_max, Cr_min, Cr_max] para las cuatro regiones.']);
94     end
95 %
96     % Asegurarse de que la matriz de puntos tenga 4 columnas
97     if size(points_matrix, 2) ~= 4
98         error(['La matriz de puntos de control debe contener exactamente 4 ' ...
99               'columnas (Cb_min, Cb_max, Cr_min, Cr_max).']);
100    end
101 %
102    % Verificar y ajustar el directorio del conjunto de datos
103    if nargin < 2
104        dataset_type = 'Training'; % Valor predeterminado: directorio "Training"
105    end
106 %
107    % Agregar sufijo "-Dataset" al directorio si es necesario
108    if ~contains(dataset_type, '-Dataset')
109        dataset_type = strcat(dataset_type, '-Dataset');
110    end
111 end
112 %
113 % Subfunción para configurar directorios
114 function [input_dir, output_dir] = setupDirectories(dataset_type)
115     % Directorio base
116     base_dir = '/Users/pol/Desktop/PIV/Laboratorio/Prog1';
117 %
118     % Directorio de entrada de imágenes
119     input_dir = fullfile(base_dir, dataset_type, 'Images');

```

```

120 % Directorio de salida de máscaras
121 output_dir = fullfile(base_dir, dataset_type, 'Masks');
123
124 % Comprobar y crear el directorio de salida si no existe
125 if ~exist(output_dir, 'dir')
126 mkdir(output_dir);
127 end
128 end
129 %-
130 % Subfunción para procesar imágenes
131 function processImages(points_matrix, points_removal_matrix, input_dir, ...
132                         output_dir, umbralApertura, processingOptions)
133
134 % Definimos las operaciones de post-procesamiento
135 shadowsRemoval = processingOptions(1);
136 filling = processingOptions(2);
137 smoothing = processingOptions(3);
138
139 % Obtener la lista de archivos de imágenes en el directorio de entrada
140 image_files = dir(fullfile(input_dir, '*.jpg'));
141
142 for n = 1:length(image_files)
143     % Leer la imagen
144     image = imread(fullfile(input_dir, image_files(n).name));
145
146     % Generar máscaras para las cuatro regiones
147     masks = generateMasks(image, points_matrix);
148
149     % Combinar las máscaras
150     combined_mask = combineMasks(masks);
151
152     if shadowsRemoval
153         % Generar máscaras para todas las regiones
154         masks_removal = generateMasks(image, points_removal_matrix);
155
156         % Combinar las máscaras
157         combined_removal_mask = combineMasks(masks_removal);
158
159         % Restarla
160         combined_mask = combined_mask & combined_removal_mask;
161     end
162
163     % Construir el nombre de archivo de salida
164     output_filename = constructOutputFilename(output_dir, image_files(n).name);
165
166     % Guardar el componente conectado más grande o la máscara combinada
167     if saveLargestConnectedComponent(combined_mask, output_filename, ...
168                                         umbralApertura, filling, smoothing)
169         continue;
170     end
171
172     % Guardar la máscara combinada si no se encontraron componentes conectados
173     imwrite(~combined_mask, output_filename);
174 end
175 end
176 %-
177 % Subfunción para generar máscaras
178 function masks = generateMasks(image, points_matrix)
179     % Utilizar algo2 para generar máscaras para las regiones
180     masks = arrayfun(@(i) algo2(image, points_matrix(i, :)), ...
181                      1:size(points_matrix, 1), 'UniformOutput', false);
182 end
183 %-
184 % Subfunción para combinar máscaras
185 function combined_mask = combineMasks(masks)
186     % Combinar las máscaras negando cualquier máscara activa
187     combined_mask = ~any(cat(3, masks{:}), 3);
188 end
189 %-
190 % Subfunción para construir nombres de archivo de salida
191 function output_filename = constructOutputFilename(output_dir, image_filename)
192     % Obtener el nombre base del archivo de imagen
193     [~, base_name, ~] = fileparts(image_filename);
194
195     % Combinar con el directorio de salida y el formato de archivo
196     output_filename = fullfile(output_dir, [base_name, '.bmp']);
197 end
198 %-
199 % Subfunción para guardar el componente conectado más grande
200 function save = saveLargestConnectedComponent(combined_mask, output_filename, ...
201                                               umbralApertura, filling, smoothing)
202     % Encontrar componentes conectados en la máscara combinada
203     cc = bwconncomp(combined_mask);
204

```

```

205 % Variable para rastrear es necesario realizar un proceso de apertura
206 apertura = false;
207
208 if cc.NumObjects > 0
209     % Calcular el número de píxeles en cada componente
210     numPixels = cellfun(@numel, cc.PixelIdxList);
211
212     % Comprobar si el componente mayor es demasiado grande
213     % *Indicar de píxeles de piel no pertenecientes a la mano*
214     if max(numPixels) > umbralApertura
215         % Si es así, realiza una erosión para reducir el tamaño del componente
216         se = strel('sphere', 10); % Esfera de radio 10
217         combined_mask = imerode(combined_mask, se);
218
219         %Encontrar las nuevas componentes
220         cc = bwconncomp(combined_mask);
221         numPixels = cellfun(@numel, cc.PixelIdxList);
222         apertura = true;
223     end
224
225 % Encontrar el índice del componente más grande
226 [~, idx] = max(numPixels);
227
228 % Crear una nueva máscara con solo el componente más grande
229 largestComponent = false(size(combined_mask));
230 largestComponent(cc.PixelIdxList{idx}) = true;
231
232 % Si se realizó una erosión previamente, realiza una dilatación
233 % para restaurar el tamaño
234 if apertura
235     largestComponent = imdilate(largestComponent, se);
236 end
237
238 mask = largestComponent;
239
240 if filling
241     % Rellenar los huecos en la máscara
242     mask = imfill(mask, 'holes');
243 end
244
245 if smoothing
246     % Suavizar los bordes de la máscara
247     mask = smoothEdges(mask);
248 end
249
250 if (filling && smoothing)
251     % Rellenar los huecos en la máscara
252     mask = imfill(mask, 'holes');
253 end
254
255 % Guardar la máscara en el directorio de salida
256 imwrite(~mask, output_filename);
257
258 save = true;
259 else
260     % No se encontraron componentes conectados, guardar la máscara original
261     imwrite(combined_mask, output_filename);
262     save = false;
263 end
264 end
265 %-----%
266 % Subfunción para suavizar los bordes por apertura
267 function smoothed_mask = smoothEdges(mask)
268     % Crear un elemento estructurante (kernel) para suavizado de bordes
269     se = strel('disk', 4); % Kernel circular de radio 4
270
271     % Aplicar dilatación seguida de erosión para suavizar los bordes
272     smoothed_mask = imerode(imdilate(mask, se), se);
273 end
274 %-----%

```

Si bien ya hemos comentado anteriormente cuales son las mejoras que hemos aplicado a la nueva función “algo3” por separado, vamos a explicar brevemente en que consiste la nueva función en su conjunto:

1. La función “algo3” toma cuatro argumentos de entrada: “points\_matrix”, “dataset\_type”, “umbralApertura” y “processingOptions”.
  - a. “points\_matrix” es una matriz que contiene puntos de control para definir regiones de piel.
  - b. “dataset\_type” es el tipo de directorio del conjunto de datos (“Training” o “Validation”).
  - c. “umbralApertura” es un umbral en número de píxeles a partir del cual las imágenes que lo superen se les aplicará un proceso de apertura para facilitar la eliminación de elementos interferentes.
  - d. “processingOptions” es un vector de booleanos que controla varias opciones de procesamiento, como la reducción de sombras, el relleno de máscaras y el suavizado de bordes.
2. La función verifica los argumentos de entrada, asegurándose de que la matriz de puntos “points\_matrix” tenga el formato correcto y ajustando el directorio del conjunto de datos.
3. Luego, configura los directorios de entrada y salida de imágenes.
4. Dependiendo de las opciones de procesamiento, la función generará máscaras de detección de piel para las imágenes en el directorio de entrada. Puede realizar una reducción de sombras si está habilitada y combina las máscaras resultantes.
5. Si se encuentra un componente conectado demasiado grande en la máscara combinada, la función realiza una apertura (erosión seguida de dilatación) para reducir su tamaño y, a continuación, guarda el componente más grande en la máscara de salida.
6. Finalmente, la función rellena y/o suaviza la máscara resultante según las opciones de postprocesamiento definidas y guarda la máscara.
7. También hay subfunciones dentro de “algo3” que ayudan en diversas tareas, como verificar argumentos, configurar directorios, procesar imágenes, generar máscaras, combinar máscaras, guardar el componente más grande y suavizar bordes, para así simplificar el código y hacerlo más versátil.

En pocas palabras, esta función realiza un proceso completo de generación de máscaras de detección de piel, con opciones de preprocesamiento y postprocesamiento, y guarda las máscaras resultantes en un directorio de salida.

Después de haber aplicado una serie de mejoras a nuestros algoritmos anteriores, decidimos desarrollar una versión mejorada del algoritmo “algo4”. En esta versión, además de calcular las métricas tradicionales (precisión, recuperación y F-Score) para evaluar el rendimiento de nuestras máscaras, hemos implementado una funcionalidad adicional que muestra gráficamente cómo estas métricas varían para cada imagen en el conjunto de datos. También proporcionando la media de estas métricas para ayudarnos a comprender mejor el impacto de los parámetros utilizados en la detección de piel y cómo afectan el rendimiento del algoritmo.

Además, se añade una pestaña adicional donde se muestra cada imagen original con su correspondiente máscara creada y una gráfica de sus valores individuales.

```
function FScore = algo4Graphics(dataset_type, createPlot)
```

```

1 function FScore = algo4Graphics(dataset_type, createPlot)
2 %-----
3 %
4 % algo4Graphics - Calcula el F-score comparando máscaras generadas y máscaras
5 % ideales y crea una gráfica.
6 %
7 %
8 %
9 % Entradas:
10 % - dataset_type: Tipo de conjunto de datos, "Training" o "Validation."
11 % - createPlot: Booleano que indica si se debe crear una gráfica.
12 %
13 %
14 %
15 % Explicación de la función:
16 % La función algo4Graphics compara las máscaras generadas por nosotros con las
17 % máscaras ideales para un conjunto de imágenes y calcula el F-score.
18 % El F-score es una métrica que combina la precisión (Precision) y la
19 % exhaustividad (Recall) para evaluar la calidad de las máscaras generadas
20 % en la detección de piel. Si createPlot es true, se crea una gráfica que
21 % muestra Precision, Recall y F-Score para cada imagen.
22 %
23 %
24 %
25 % Ejemplo de uso:
26 % FScore = algo4Graphics('Validation', true); % Calcular el F-score para el
27 % % conjunto de validación y
28 % % crear una gráfica
29 %
30 %
31 %
32 % Salida:
33 % - FScore: Valor del F-score que indica la calidad de las máscaras generadas.
34 %
35 %
36 %
37 % Verificar si se proporcionó el tipo de conjunto de datos
38 if nargin < 1
39 % Valor predeterminado del tipo de conjunto de datos
40 dataset_type = 'Training'; % Directorio "Training-Dataset" por defecto
41 end
42 %
43 % Verificar si se debe crear una gráfica
44 if nargin < 2
45 createPlot = false; % Valor predeterminado: no crear la gráfica
46 end
47 %
48 % Agregar "-Dataset" al nombre del directorio
49 dataset_dir = strcat(dataset_type, '-Dataset');
```

```

51 % Directorios de entrada
52 input_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
53 dataset_dir, 'Images');
54 mask_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
55 dataset_dir, 'Masks');
56 ideal_mask_dir = fullfile('/Users/pol/Desktop/PIV/Laboratorio/Prog1', ...
57 dataset_dir, 'Masks-Ideal');
58
59 % Información de las imágenes en el directorio de entrada
60 lista_im = dir(fullfile(input_dir, '*.jpg'));
61
62 % Variables para el cálculo de F-score
63 vect_TP = zeros(1, length(lista_im));
64 vect_T = zeros(1, length(lista_im));
65 vect_P = zeros(1, length(lista_im));
66
67 if createPlot
68 % Crear una figura
69 figure('Position', [100, 100, 1000, 400]);
70
71 % Agregar un control deslizante (slider) para seleccionar la imagen
72 slider = uicontrol('Style', 'slider', 'Min', 1, 'Max', length(lista_im), ...
73 'Value', 1, 'SliderStep', [1/(length(lista_im)-1) 1/(length(lista_im)-1)]);
74 'Position', [100 20 300 20], 'Callback', @sliderCallback);
75
76 % Agregar un texto para mostrar el valor del slider
77 text = uicontrol('Style', 'text', 'Position', [420 20 40 20], ...
78 'FontWeight', 'bold');
79 end
80
81 % Función de devolución de llamada para el slider
82 function sliderCallback(hObject, ~)
83 slider_value = round(hObject.Value); % Obtener el valor del slider
84
85 % Actualizar el texto con el valor del slider
86 set(text, 'String', num2str(slider_value));
87
88 % Leer la imagen correspondiente al valor del slider
89 selected_image_name = lista_im(slider_value).name;
90 mask_nuestra = imread(fullfile(mask_dir, strrep(selected_image_name, ...
91 'jpg', 'bmp')));
92
93 % Mostrar la imagen y la máscara en el subplot izquierdo
94 subplot(1, 3, 1);
95 imshow(mask_nuestra);
96 title(['Máscara de la imagen ' num2str(slider_value)]);
97
98 % También puedes mostrar la imagen original si lo deseas
99 img_original = imread(fullfile(input_dir, selected_image_name));
100 subplot(1, 3, 2);
101 imshow(img_original);
102 title(['Imagen original ' num2str(slider_value)]);
103
104 % Calcular Precision, Recall y F-Score para la imagen seleccionada
105 TP_im = vect_TP(slider_value);
106 P_im = vect_P(slider_value);
107 T_im = vect_T(slider_value);
108 Precision_im = TP_im / P_im;
109 Recall_im = TP_im / T_im;
110 FScore_im = 2 * Precision_im * Recall_im / (Precision_im + Recall_im);
111
112 % Crear la gráfica en el subplot derecho
113 subplot(1, 3, 3);
114 bar([Precision_im, Recall_im, FScore_im]);
115 title(['Precision, Recall, F-Score para la imagen ' num2str(slider_value)]);
116 set(gca, 'XTickLabel', {'Precision', 'Recall', 'F-Score'});
117 end
118
119 % Por cada imagen en el directorio de entrada
120 for n = 1:length(lista_im)
121 % Leer la máscara generada por nosotros y la máscara ideal
122 mask_nuestra = imread(fullfile(mask_dir, strrep(lista_im(n).name, ...
123 'jpg', 'bmp')));
124 mask_ideal = ~imread(fullfile(ideal_mask_dir, strrep(lista_im(n).name, ...
125 'jpg', 'bmp')));
126
127 % Calcular TP, T y P utilizando operaciones vectorizadas
128 TP = sum(mask_nuestra(:) & mask_ideal(:));
129 P = sum(mask_nuestra(:));
130 T = sum(mask_ideal(:));
131
132 % Almacenar los resultados en vectores
133 vect_TP(n) = TP;

```

```

134     vect_T(n) = T;
135     vect_P(n) = P;
136 end
137
138
139 % Calcular TP, T y P totales
140 total_TP = sum(vect_TP);
141 total_T = sum(vect_T);
142 total_P = sum(vect_P);
143
144 % Calcular Precisión y Recall
145 Precision = total_TP / total_P;
146 Recall = total_TP / total_T;
147
148 % Calcular F-Score
149 FScore = 2 * Precision * Recall / (Precision + Recall);
150
151
152 % Crear la gráfica si createPlot es true
153 if createPlot
154     sliderCallback(slider, []);
155
156     vect_precision = zeros(1, length(lista_im));
157     vect_recall = zeros(1, length(lista_im));
158     vect_fScore = zeros(1, length(lista_im));
159
160     for n = 1:length(lista_im)
161         % Calcular Precision, Recall y F-Score para cada imagen
162         vect_precision(n) = vect_TP(n) / vect_P(n);
163         vect_recall(n) = vect_TP(n) / vect_T(n);
164         vect_fScore(n) = 2 * vect_precision(n) * vect_recall(n) / ...
165             (vect_precision(n) + vect_recall(n));
166     end
167
168 % Crear la gráfica con Precision, Recall y F-Score
169 figure;
170 x = 1:length(lista_im);
171
172 % Define la variable que representa el valor deseado
173 % Representar la precisión
174 subplot(3, 1, 1);
175 plot(x, vect_precision, 'r');
176 title('Gráfica de Precision, Recall y F-Score por Número de Imagen');
177 xlabel('Número de Imagen');
178 ylabel('Precision');
179 legend('Precision');
180 h1 = refline(0, Precision);
181 h1.DisplayName = 'mean';
182
183 % Representar el recall
184 subplot(3, 1, 2);
185 plot(x, vect_recall, 'g');
186 xlabel('Número de Imagen');
187 ylabel('Recall');
188 legend('Recall');
189 h2 = refline(0, Recall);
190 h2.DisplayName = 'mean';
191
192 % Representar el F-Score
193 subplot(3, 1, 3);
194 plot(x, vect_fScore, 'b');
195 xlabel('Número de Imagen');
196 ylabel('F-Score');
197 legend('F-Score');
198 h3 = refline(0, FScore);
199 h3.DisplayName = 'mean';
200 end
201 end

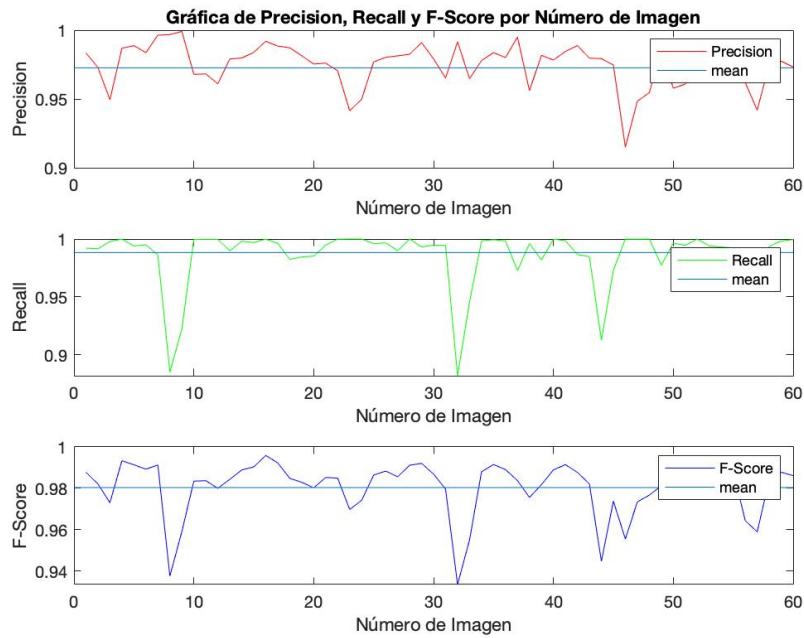
```

A continuación, se describe el funcionamiento del código:

1. Se verifica si se proporciona el tipo de conjunto de datos y si se debe crear una gráfica.
2. Se establecen los directorios de entrada para las imágenes, las máscaras generadas por nosotros y las máscaras ideales.

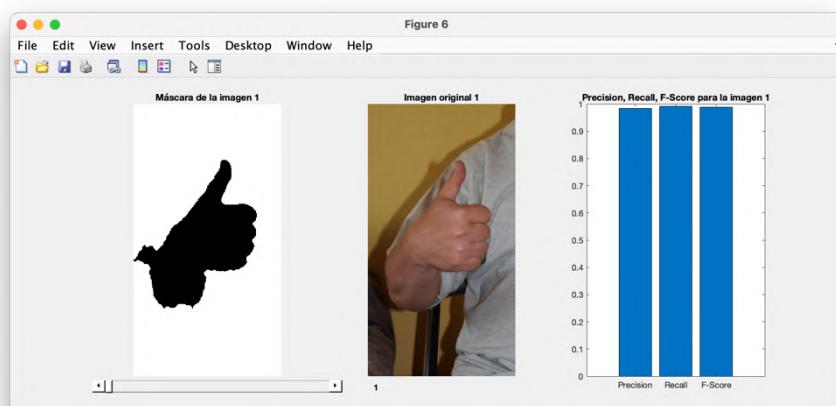
3. Se crea una figura que contendrá la gráfica y los resultados. Además, se agrega un control deslizante que permite seleccionar una imagen específica para visualizar sus métricas.
4. Se define una función de devolución de llamada para el control deslizante que actualiza la visualización cuando se selecciona una imagen diferente.
5. Se itera a través de todas las imágenes en el conjunto de datos. Para cada imagen:
  - a. Se leen la máscara generada por nosotros y la máscara ideal.
  - b. Se calculan los valores de Verdaderos Positivos (TP), Positivos (P) y Verdaderos (T) utilizando operaciones vectorizadas, lo que mejora la eficiencia del cálculo.
  - c. Los resultados de TP, P y T se almacenan en vectores para su posterior uso.
6. Se calculan los valores totales de TP, P y T para todas las imágenes.
7. Se calculan las métricas de Precisión y Recuperación (Precision y Recall) para el conjunto de datos completo.
8. Se calcula el F-Score utilizando las métricas de Precisión y Exhaustividad.
9. Se llama a la función de devolución de llamada del control deslizante para mostrar inicialmente la primera imagen y sus métricas.
10. Si se establece la opción “createPlot” como verdadera, se crea una gráfica adicional que muestra cómo varían las métricas (Precision, Recall y F-Score) para cada imagen en función de su posición en el conjunto de datos. También se representa la media de estas métricas en la gráfica.

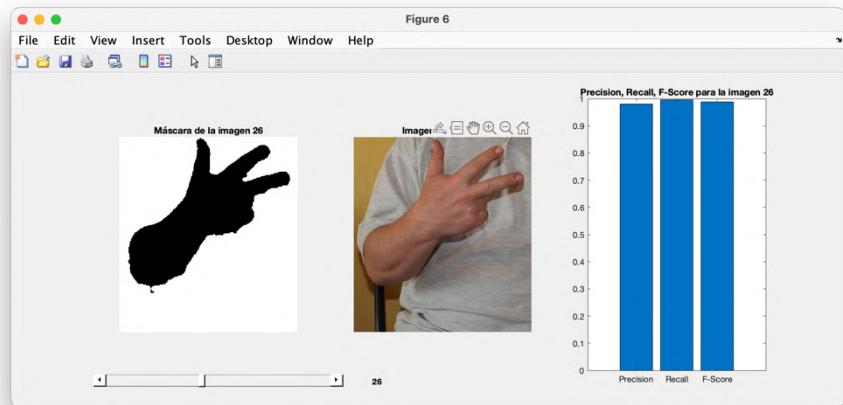
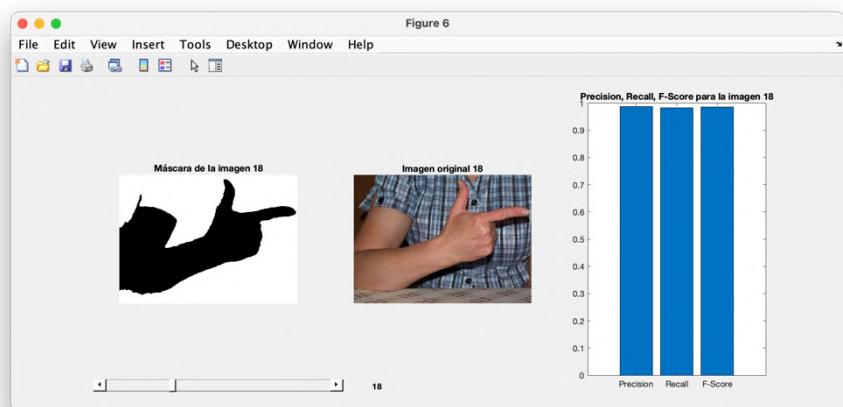
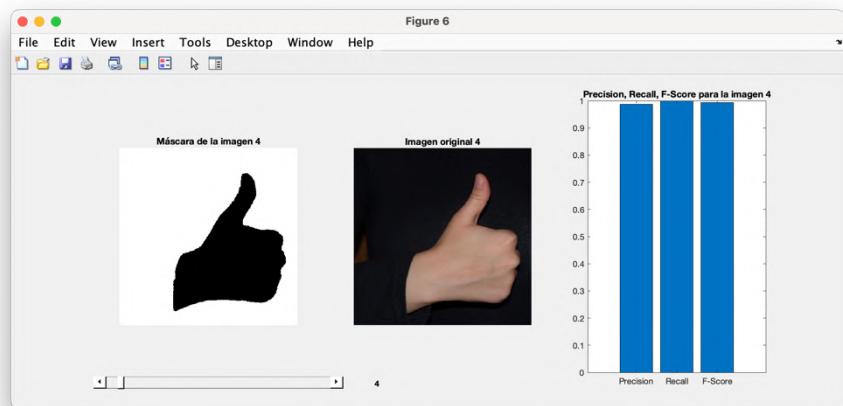
A continuación, se muestran ejemplos de ejecución de la función en el conjunto de validación y con `createPlot` en “true”:

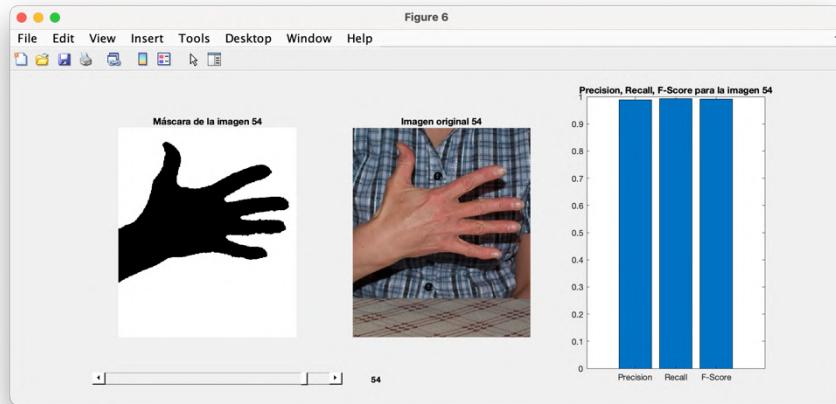


### Subgráficas de Precision, Recall y F-Score

Y a continuación ejemplos de la interfaz de visualización de máscaras individuales proporcionada por “algo4”:







Y con todos estos algoritmos ya estamos listos para definir un programa principal “main” de nuestro sistema.

Inicialmente decidimos diseñar un simple script donde ejecutar todos nuestros algoritmos en un único sistema, sin embargo, ese script inicialmente denominado “Prog1” se vio obligado a evolucionar. Al haber aplicado tanta funcionalidad debíamos adaptar nuestro sistema al usuario, y crear una interfaz simple que permitiera modificar y adaptar todos estos parámetros de forma simple e intuitiva. A continuación, mostramos primero el script preliminar del que nace la interfaz y luego mostraremos el código de la susodicha interfaz de usuario junto con su ejecución.

```

1 % Definir los valores de los umbrales para cada región de piel
2 Cb_min = 114; Cb_max = 149;
3 Cr_min = 102; Cr_max = 137;
4
5 Cb_min2 = 108; Cb_max2 = 115;
6 Cr_min2 = 155; Cr_max2 = 168;
7
8 Cb_min3 = 95; Cb_max3 = 111;
9 Cr_min3 = 139; Cr_max3 = 146;
10
11 Cb_min4 = 111; Cb_max4 = 118;
12 Cr_min4 = 137; Cr_max4 = 138;
13
14 Cb_min5 = 112; Cb_max5 = 114;
15 Cr_min5 = 140; Cr_max5 = 142;
16
17
18 % Opciones de procesamiento
19 dataset_type = 'Training'; % Directorio del conjunto de datos
20 umbralApertura = 98000; % Umbral para el proceso de apertura
21 shadowsRemoval = true; % Habilitar la reducción de sombras
22 filling = true; % Habilitar el relleno de la máscara
23 smoothing = true; % Habilitar el suavizado de bordes
24 createPlot = true; % Habilitar gráficas de las métricas
25
26
27 % Definir los vectores de puntos de control para cada región
28 points = [Cb_min, Cb_max, Cr_min, Cr_max];
29 points2 = [Cb_min2, Cb_max2, Cr_min2, Cr_max2];
30 points3 = [Cb_min3, Cb_max3, Cr_min3, Cr_max3];
31 points4 = [Cb_min4, Cb_max4, Cr_min4, Cr_max4];
32 points5 = [Cb_min5, Cb_max5, Cr_min5, Cr_max5];
33
34 % Crear un vector de opciones de procesamiento
35 processingOptions = [shadowsRemoval, filling, smoothing];
36
37 % Definir la matriz points_matrix concatenando los vectores verticalmente
38 points_matrix = [points; points2; points3; points4; points5];
39
40 % Mostrar histogramas de las regiones de piel utilizadas (opcional)
41 %algo1(points, dataset_type)
42 %algo1(points2, dataset_type)
43 %algo1(points3, dataset_type)
44 %algo1(points4, dataset_type)
45 %algo1(points5, dataset_type)
46
47 % Llamar a la función algo3 para generar máscaras de detección de piel
48 algo3(points_matrix, dataset_type, umbralApertura, processingOptions)
49
50 % Llama a la función algo4 y capture el F-Score
51 FScore = algo4Graphics(dataset_type, false);
52
53 % Mostrar el valor del F-Score en la consola
54 disp(['El valor del F-Score es: ' num2str(FScore)]);

```

El main del sistema comienza definiendo los umbrales de color en el espacio YCbCr para cada todas las regiones de piel. Luego, se establecen las opciones de procesamiento, que incluyen la habilitación de la reducción de sombras, el relleno de la máscara y el suavizado de bordes. También se configura la creación de gráficas para visualizar métricas.

A continuación, se definen vectores de puntos de control para cada región de piel y se crea una matriz que concatena estos vectores. Se podrían mostrar histogramas de estas regiones de piel, pero la opción está comentada para reducir el tiempo de procesado durante las pruebas.

Luego, se llama a la función “algo3” para generar máscaras de detección de piel, pasando los valores de umbral y las opciones de procesamiento. Posteriormente, se llama a la función “algo4Graphics” para calcular y

visualizar el F-Score, utilizando la opción “false” para evitar la creación de gráficas. Finalmente, se muestra el valor del F-Score en la consola.

Este sería en resumen el funcionamiento de nuestro sistema, no obstante, a nivel de usuario el diseño de una interfaz sería una gran mejora. A continuación, se muestra interfaz diseñada:

```

1 function SistemaDeteccion()
2     % Valores predeterminados de regiones de detección
3     Cb_min = 114; Cb_max = 149;
4     Cr_min = 102; Cr_max = 137;
5
6     Cb_min2 = 108; Cb_max2 = 115;
7     Cr_min2 = 155; Cr_max2 = 168;
8
9     Cb_min3 = 95; Cb_max3 = 111;
10    Cr_min3 = 139; Cr_max3 = 146;
11
12    Cb_min4 = 111; Cb_max4 = 118;
13    Cr_min4 = 137; Cr_max4 = 138;
14
15    Cb_min5 = 112; Cb_max5 = 114;
16    Cr_min5 = 140; Cr_max5 = 142;
17
18    umbralApertura = 98000;           % Umbral para el proceso de apertura
19
20    % Processing options
21    shadowsRemoval = true;           % Habilitar la reducción de sombras
22    filling = true;                 % Habilitar el relleno de la máscara
23    smoothing = true;               % Habilitar el suavizado de bordes
24
25    % Análisis
26    createPlot = false;              % Habilitar gráficas de las métricas
27
28
29    % Crear la ventana principal
30    figure('Name', 'Sistema de detección', 'NumberTitle', 'off', ...
31        'Position', [350, 270, 680, 440], 'Toolbar', 'none', ...
32        'MenuBar', 'none', 'resize', 'off');
33
34    % Títulos principales
35    uicontrol('Style', 'text', 'Position', [95, 390, 200, 20], 'String', ...
36        'Secciones de detección', 'FontWeight', 'bold', ...
37        'FontSize', 14, 'ForegroundColor', [0, 0, 0.8], ...
38        'FontName', 'underline');
39    uicontrol('Style', 'text', 'Position', [395, 360, 60, 20], 'String', ...
40        'Ajustes', 'FontWeight', 'bold', ...
41        'FontSize', 14, 'ForegroundColor', [0.5, 0.2, 0.2]);
42
43
44    % Crear 20 cuadros de edición para los valores
45    editBoxes = zeros(1, 20);
46    labels = {'Cb_min', 'Cb_max', 'Cr_min', 'Cr_max', ...
47        'Cb_min2', 'Cb_max2', 'Cr_min2', 'Cr_max2', ...
48        'Cb_min3', 'Cb_max3', 'Cr_min3', 'Cr_max3', ...
49        'Cb_min4', 'Cb_max4', 'Cr_min4', 'Cr_max4', ...
50        'Cb_min5', 'Cb_max5', 'Cr_min5', 'Cr_max5'};
51
52    row = 320;
53    col = 60;
54    for i = 1:20
55        editBoxes(i) = uicontrol('Style', 'edit', 'Position', ...
56            [col, row, 50, 20], 'String', ...
57            num2str(eval(labels{i})));
58        col = col + 70;
59        if mod(i, 4) == 0
60            col = 60;
61            row = row - 55;
62        end
63    end
64
65    % Agregar etiquetas de cada sección
66    uicontrol('Style', 'text', 'Position', [140, 341, 100, 20], 'String', ...
67        'Sección 1', 'FontWeight', 'bold');
68    uicontrol('Style', 'text', 'Position', [140, 286, 100, 20], 'String', ...
69        'Sección 2', 'FontWeight', 'bold');
70    uicontrol('Style', 'text', 'Position', [140, 231, 100, 20], 'String', ...
71        'Sección 3', 'FontWeight', 'bold');

```

```

72 uicontrol('Style', 'text', 'Position', [140, 176, 100, 20], 'String', ...
73     'Sección 4', 'FontWeight', 'bold');
74 uicontrol('Style', 'text', 'Position', [140, 121, 100, 20], 'String', ...
75     'Sección 5', 'FontWeight', 'bold');
76
77 % Agregar etiquetas que indique la variable de cada casilla
78 uicontrol('Style', 'text', 'Position', [35, 365, 100, 20], 'String', 'Cb_min');
79 uicontrol('Style', 'text', 'Position', [105, 365, 100, 20], 'String', 'Cb_max');
80 uicontrol('Style', 'text', 'Position', [175, 365, 100, 20], 'String', 'Cr_min');
81 uicontrol('Style', 'text', 'Position', [245, 365, 100, 20], 'String', 'Cr_max');
82
83 %-
84
85 % Menú desplegable para seleccionar dataset_type
86 datasetTypeMenu = uicontrol('Style', 'popupmenu', 'String', ...
87     {'Training', 'Validation'}, 'Position', ...
88     [477, 226, 100, 120]);
89
90 % Etiqueta para el banco de datos
91 uicontrol('Style', 'text', 'Position', [395, 320, 78, 25], ...
92     'String', 'Banco de datos:');
93
94 %-
95
96 % Recuadro para imprimir el valor de F-Score
97 fscoreText = uicontrol('Style', 'text', 'Position', [430, -90, 100, 170], ...
98     'String', 'F-Score: ', 'FontWeight', 'bold');
99
100 % Botón para actualizar el valor de F-Score
101 uicontrol('Style', 'pushbutton', 'String', 'Actualizar F-Score', ...
102     'Position', [50, 50, 120, 25], 'Callback', @updateFScore);
103
104 %-
105
106 % Botón para generar histogramas
107 uicontrol('Style', 'pushbutton', 'String', 'Generar Histogramas', ...
108     'Position', [200, 50, 150, 25], 'Callback', @generarHistogramas);
109
110 %
111
112 % Agregar un cuadro de edición para umbralApertura
113 umbralAperturaEdit = uicontrol('Style', 'edit', 'Position', ...
114     [555, 284, 70, 25], 'String', ...
115     num2str(umbralApertura));
116
117 % Etiqueta para el cuadro de edición de umbralApertura
118 uicontrol('Style', 'text', 'Position', [395, 280, 150, 25], 'String', ...
119     'Umbral para la Apertura [#pixels]:');
120
121 %
122
123 % Etiqueta para el cuadro de procesado
124 uicontrol('Style', 'text', 'Position', [395, 235, 140, 25], 'String', ...
125     'Operaciones de procesado: ', 'FontWeight', 'bold');
126
127 % Etiqueta para el cuadro de métricas
128 uicontrol('Style', 'text', 'Position', [385, 130, 180, 25], 'String', ...
129     'Análisis de máscaras y métricas: ', 'FontWeight', 'bold');
130
131 % Crear una casillas de verificación
132 shadowsRemovalCheckbox = uicontrol('Style', 'checkbox', 'Position', ...
133     [415, 220, 120, 20], 'String', ...
134     'Eliminar Sombras', 'Value', ...
135     shadowsRemoval, 'Callback', ...
136     @toggleShadowsRemoval);
137
138 fillingCheckbox = uicontrol('Style', 'checkbox', 'Position', ...
139     [415, 200, 120, 20], 'String', ...
140     'Habilitar Rellenado', 'Value', ...
141     filling, 'Callback', @toggleFilling);
142
143 smoothingCheckbox = uicontrol('Style', 'checkbox', 'Position', ...
144     [415, 180, 120, 20], 'String', ...
145     'Habilitar Suavizado', 'Value', ...
146     smoothing, 'Callback', @toggleSmoothing);
147
148 createPlotCheckbox = uicontrol('Style', 'checkbox', 'Position', ...
149     [415, 115, 140, 20], 'String', ...
150     'Habilitar Gráficas Métricas', ...
151     'Value', createPlot, 'Callback', ...
152     @toggleCreatePlot);
153
154 %
155
156 function updateFScore(~, ~)
157     % Recopilar el valor de umbralApertura del cuadro de edición
158     umbralApertura = str2double(get(umbralAperturaEdit, 'String'));
159

```

```

160 % Recopilar los valores de los cuadros de edición
161 values = getValues;
162
163 % Obtener el dataset_type seleccionado
164 datasetType = datasetTypeMenu.String{datasetTypeMenu.Value};
165
166 % Llamar a algo3 con los valores recopilados y el dataset_type
167 % Crear matrices points, points2, points3, points4 y points5
168 points = (values(1:4));
169 points2 = (values(5:8));
170 points3 = (values(9:12));
171 points4 = (values(13:16));
172 points5 = (values(17:20));
173
174 % Definir la matriz points_matrix concatenando los vectores verticalmente
175 points_matrix = [points; points2; points3; points4; points5];
176
177 % Llamar a algo3 con los valores recopilados y el dataset_type
178 algo3(points_matrix, datasetType, umbralApertura, ...
179 [shadowsRemoval, filling, smoothing]);
180
181 % Llamar a algo4 y capturar el F-Score
182 FScore = algo4Graphics(datasetType, createPlot);
183
184 % Actualizar el valor de F-Score en pantalla
185 set(fscoreText, 'String', ['F-Score: ', num2str(FScore)]);
186 end
187
188 function generarHistogramas(~, ~)
189 % Recopilar los valores de los cuadros de edición
190 values = getValues;
191
192 % Llamar a algo1 con los valores de la sección 1
193 algo1(values(1:4), datasetTypeMenu.String{datasetTypeMenu.Value});
194
195 % Llamar a algo1 con los valores de la sección 2
196 algo1(values(5:8), datasetTypeMenu.String{datasetTypeMenu.Value});
197
198 % Llamar a algo1 con los valores de la sección 3
199 algo1(values(9:12), datasetTypeMenu.String{datasetTypeMenu.Value});
200
201 % Llamar a algo1 con los valores de la sección 4
202 algo1(values(13:16), datasetTypeMenu.String{datasetTypeMenu.Value});
203
204 % Llamar a algo1 con los valores de la sección 5
205 algo1(values(17:20), datasetTypeMenu.String{datasetTypeMenu.Value});
206
207 end
208
209 function toggleShadowsRemoval(~, ~)
210 % Cambiar el valor de shadowsRemoval según el estado de la
211 % casilla de verificación
212 shadowsRemoval = get(shadowsRemovalCheckbox, 'Value');
213 end
214
215 function toggleFilling(~, ~)
216 % Cambiar el valor de filling según el estado de la
217 % casilla de verificación
218 filling = get(fillingCheckbox, 'Value');
219 end
220
221 function toggleSmoothing(~, ~)
222 % Cambiar el valor de smoothing según el estado de la
223 % casilla de verificación
224 smoothing = get(smoothingCheckbox, 'Value');
225 end
226
227 function toggleCreatePlot(~, ~)
228 % Cambiar el valor de createPlot según el estado de la
229 % casilla de verificación
230 createPlot = get(createPlotCheckbox, 'Value');
231 end
232
233 function values = getValues(~, ~)
234 % Recopilar los valores de los cuadros de edición
235 values = zeros(1, 20);
236 for i = 1:20
237     values(i) = str2double(get(editBoxes(i), 'String'));
238     if isnan(values(i))
239         msgbox(['Por favor, ingrese valores numéricos válidos en ' ...
240                 'todos los cuadros de edición.', 'Error', 'error']);
241         return;
242     end
243 end
244 end
245 end

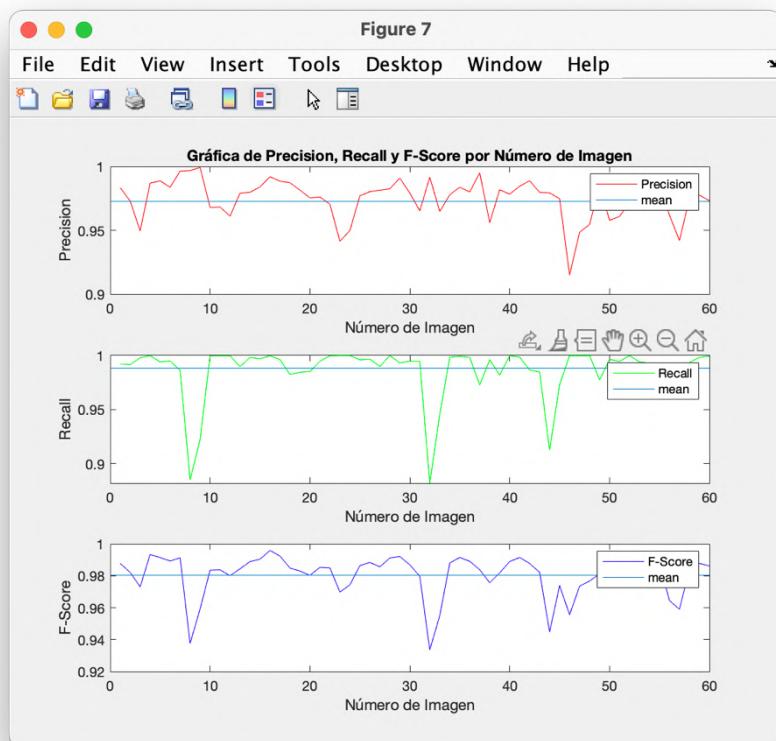
```

Hemos implementado una interfaz gráfica que permite a los usuarios interactuar con el sistema de detección de piel de una manera más intuitiva. La interfaz consta de elementos visuales que permiten configurar y ejecutar el sistema de detección de piel. La interfaz tiene el siguiente aspecto:

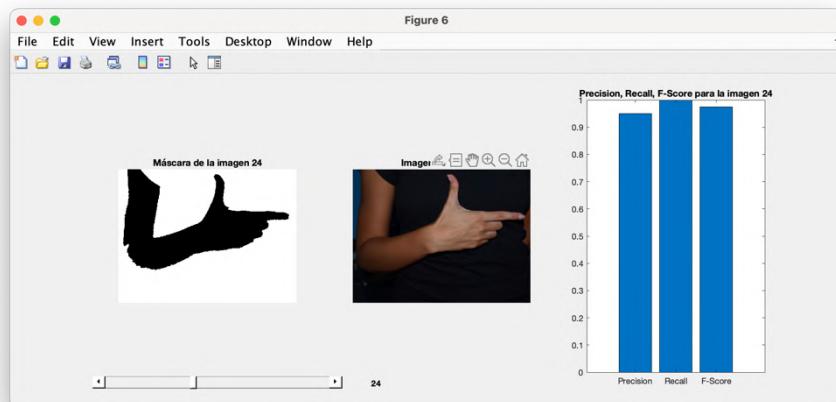


A continuación, se describe brevemente la estructura de la interfaz:

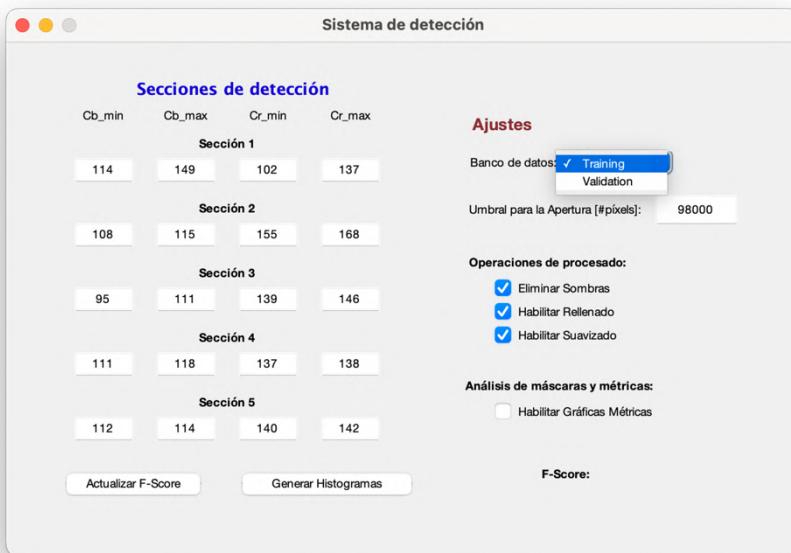
1. **Valores Predeterminados de Regiones de Detección:** La interfaz dispone de 5 regiones de detección editables por el usuario. Se establecen iniciales unos valores predeterminados para las regiones de detección de piel en el espacio de color YCbCr. Estos valores han sido los determinados como óptimos a partir de una búsqueda exhaustiva de los píxeles de pies en el conjunto de datos de entrenamientos.
2. **Gráficas de métricas y máscaras:** Si el usuario decide habilitar las gráficas métricas obtendrá acceso a las siguientes secciones:
  - a. Un conjunto de gráficas de las métricas Precision, Recall y F-Score de cada imagen junto con su media



- b. Una pestaña donde visualizar y seleccionar con un cursor (slider) cada una de las imágenes originales junto con las máscaras creadas. Mostrando además las gráficas con las métricas concretas de esa máscara.

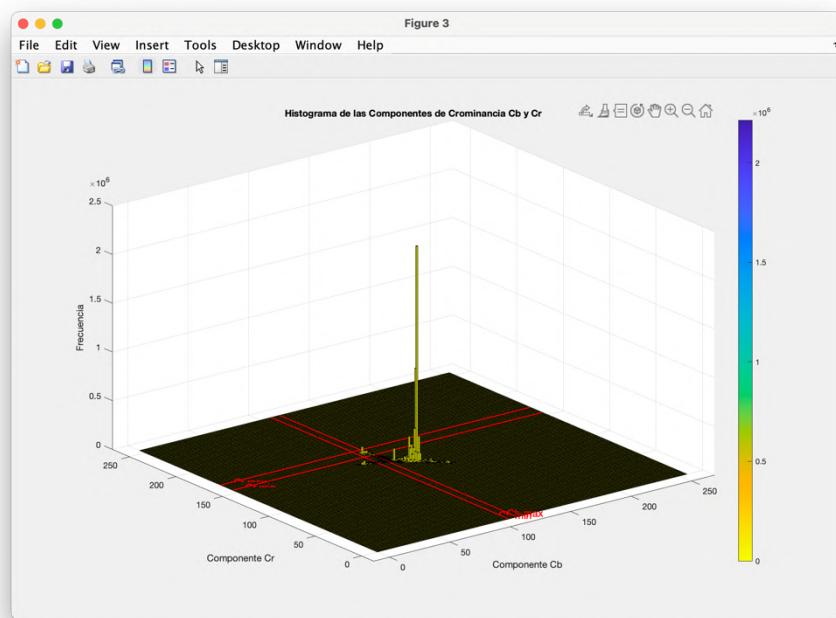
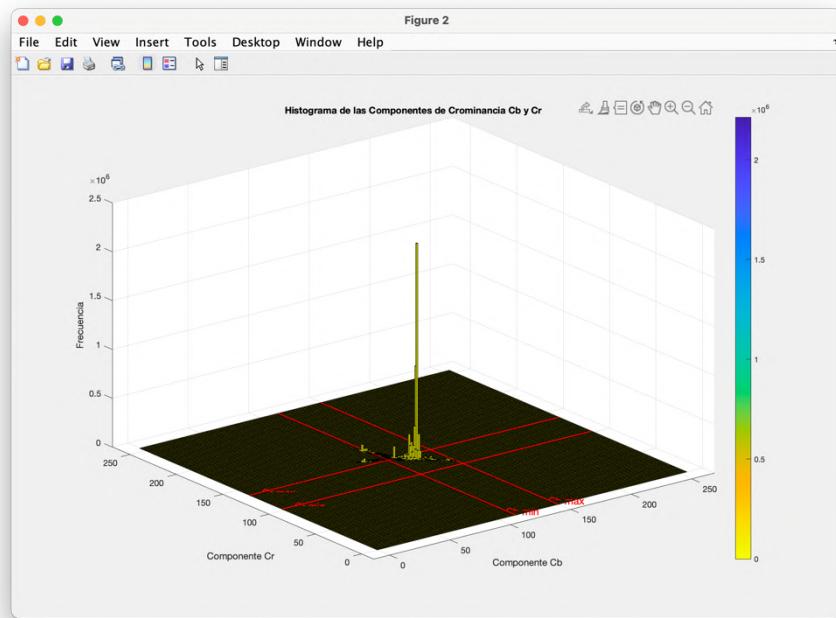


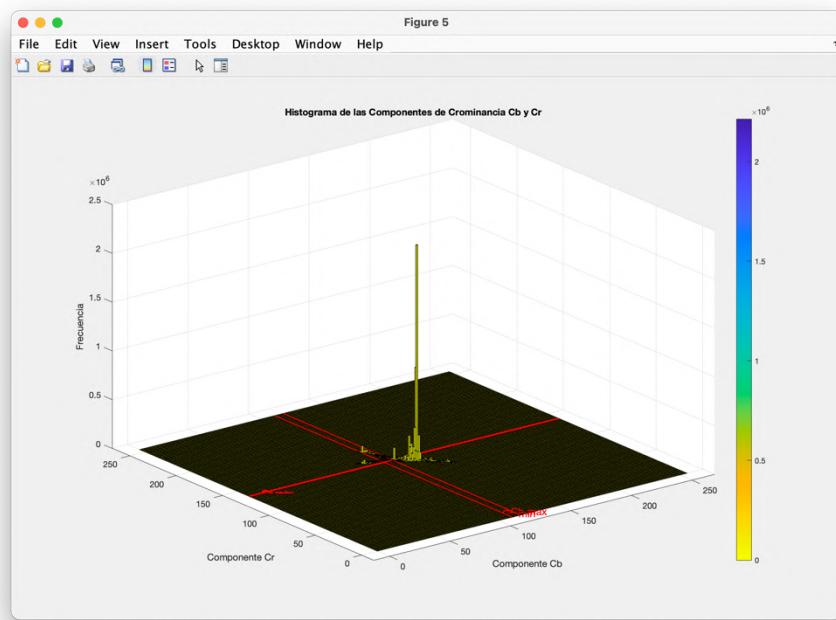
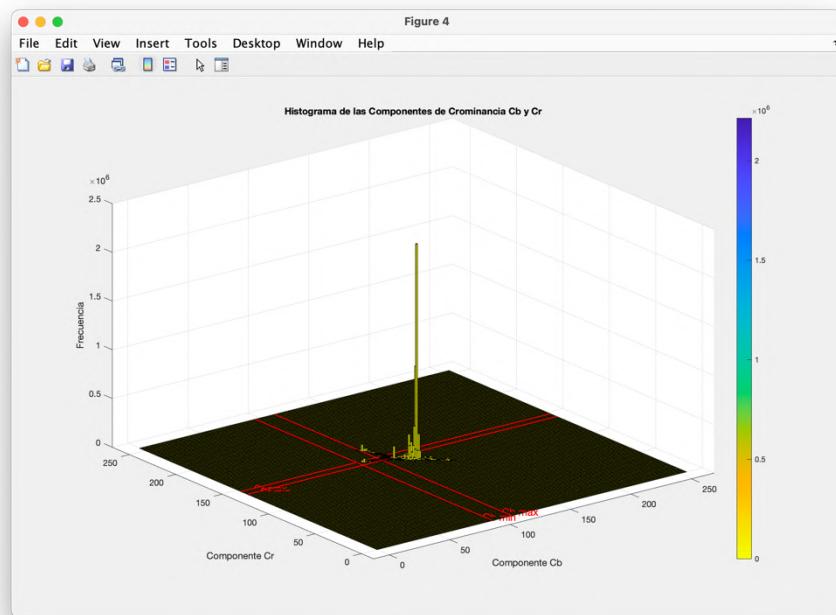
3. **Selección del Conjunto de Datos:** Se proporciona un menú desplegable que permite a los usuarios elegir entre el conjunto de datos de entrenamiento y el conjunto de datos de validación.

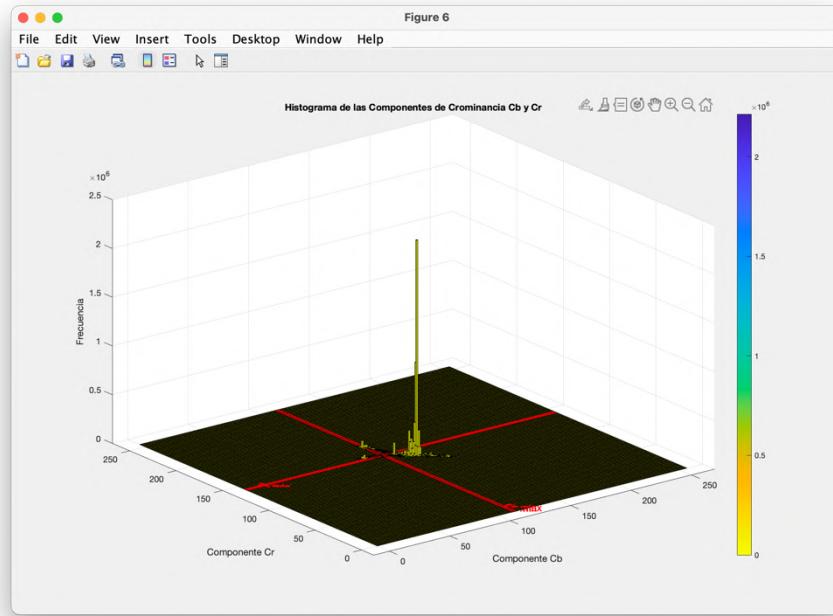


4. **Valor de Umbral para la Apertura:** Los usuarios pueden especificar el valor en número de píxeles del umbral a partir del cual las imágenes sufrirán un proceso de apertura que facilite la eliminación de componentes interferentes en las máscaras.
5. **Operaciones de Procesado:** Los usuarios pueden habilitar o deshabilitar la eliminación de sombras, el relleno de máscaras y el suavizado de bordes mediante casillas de verificación.
6. **Generar Histogramas:** Se incluye un botón que permite a los usuarios generar histogramas donde se especifiquen todas las regiones de detección de piel.

Ordenados por secciones de la 1 a la 5:







7. **Actualizar F-Score:** Los usuarios pueden hacer clic en un botón para actualizar el valor del F-Score y visualizarlo en la interfaz gráfica.
8. **Cuadros de Edición:** Los usuarios pueden ingresar valores numéricos para ajustar las regiones de detección de piel si así lo desean. Hay un total de 20 cuadros de edición, 4 puntos por región, para configurar los umbrales de detección.

La interfaz diseñada proporciona a los usuarios una forma conveniente de configurar y ejecutar el sistema de detección de piel, así como de analizar el rendimiento a través del F-Score y gráficas opcionales de métricas. Además, permite un accesible análisis modificando los parámetros y observando los resultados. Lo que en conjunto hace que el proceso de ajuste y evaluación del algoritmo sea más accesible y amigable para los usuarios.

Con esto concluimos la exposición del desarrollo de nuestro sistema. En esta sección, hemos proporcionado una narración exhaustiva de nuestra situación inicial, detallando los avances realizados, los razonamientos subyacentes que respaldan nuestras decisiones y sus correspondientes justificaciones. Tras haber explicado este proceso, hemos logrado la implementación de un sistema plenamente funcional, optimizado al máximo de nuestras capacidades.

Nuestro sistema se ha construido a partir de la información obtenida del conjunto de entrenamiento. El siguiente paso implica la presentación y evaluación de los resultados que genera nuestro sistema. Para ello, lo probaremos no solo con el conjunto de entrenamiento con el que ha sido entrenado sino también con un conjunto de datos de validación del cual no hemos extraído ninguna información previa. De esta manera, comprobaremos la eficacia de nuestro sistema en un entorno de imágenes desconocidas, lo que nos permitirá validar su versatilidad y adaptabilidad a diversos conjuntos de imágenes. Dicho esto, procederemos a la presentación y el análisis de los resultados obtenidos.

### c. Presentación de los resultados obtenidos:

- **F-Score en el conjunto de validación y en el conjunto entregado para la evaluación final (criterio 1)**

En el transcurso de este documento, hemos detallado minuciosamente el desarrollo de nuestro sistema de detección de manos. Hemos explorado las diversas mejoras y algoritmos que diseñamos y hemos evaluado su desempeño en el conjunto de entrenamiento. Examinar los resultados en este conjunto es crucial para determinar la eficiencia de nuestro sistema, sin embargo, la verdadera prueba de la capacidad de nuestro sistema radica en su rendimiento en conjuntos de datos de los cuales no tenemos información previa. Un sistema robusto debe ser capaz de funcionar eficazmente en cualquier conjunto de imágenes, basándose únicamente en el conocimiento adquirido durante el entrenamiento.

Para evaluar de manera integral el rendimiento de nuestro sistema, utilizaremos dos conjuntos de datos nuevos para nosotros, "Validation" y "Test". Estos conjuntos son ajenos a nuestro sistema y desempeñan un papel crucial en la evaluación de su eficiencia. Los resultados de F-Score obtenidos en estos conjuntos definirán de forma clara la eficiencia de nuestros algoritmos.

En esta sección, nos centraremos exclusivamente en la métrica F-Score como indicador clave de rendimiento. En secciones posteriores, abordaremos el análisis del tiempo de cálculo del sistema y profundizaremos en la evaluación detallada de los resultados obtenidos.

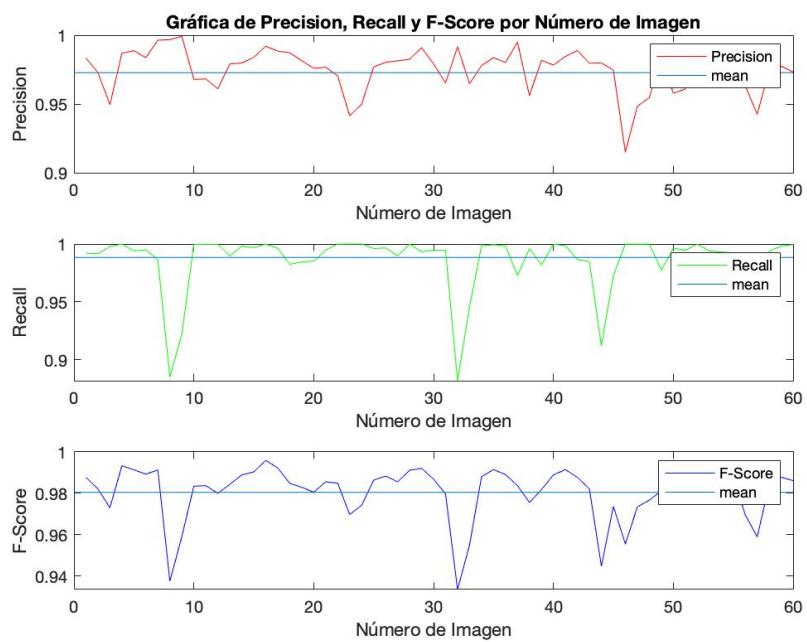
Para calcular el F-Score, utilizaremos dos enfoques diferentes. En primer lugar, aplicaremos nuestro propio método de cálculo, basado en los resultados obtenidos con nuestro sistema. Luego, validaremos estos resultados con la ayuda del sistema de verificación de máscaras proporcionado por el profesorado. Al ser un sistema externo y objetivo, esta validación nos permitirá confirmar la precisión de nuestros cálculos. Con esta doble verificación, obtendremos una comprensión completa del rendimiento de nuestro sistema y su capacidad para cumplir con los criterios establecidos.

Para comenzar, analicemos las métricas en el conjunto de entrenamiento:

# Training Dataset

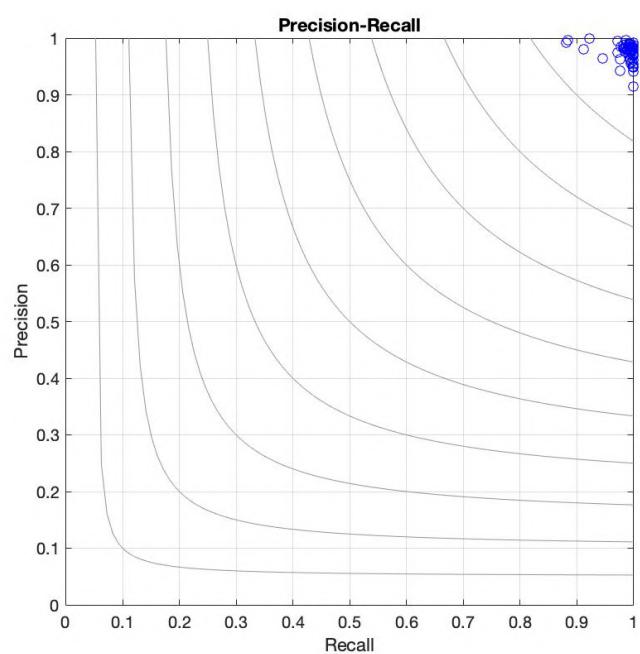
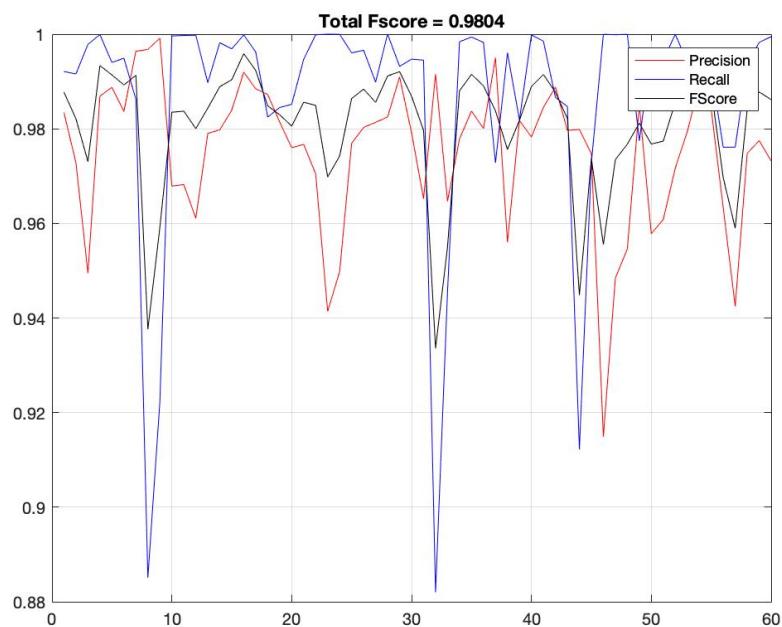
## 1. Sistema propio:

Si aplicamos nuestro propio sistema, los resultados son los siguientes:



## 2. Sistema profesorado:

Si aplicamos la herramienta de comprobación proporcionada por el profesorado, obtenemos:



### **3. Conclusiones:**

En el conjunto de entrenamiento, nuestro sistema y la herramienta del profesorado proporcionan resultados extremadamente similares, con solo una diferencia mínima en el valor del F-Score (0.00008). Una diferencia insignificante, probablemente debida a errores de truncamiento o redondeo en los cálculos.

Esta similitud de resultados no hace más que verificar la precisión de nuestro sistema. En este contexto, consideramos que tanto nuestro valor de F-Score como el proporcionado por el profesorado son altamente satisfactorios.

Un F-Score cercano a 0.9805 indica una detección de piel eficaz y precisa en las imágenes del conjunto de entrenamiento. A priori nuestro sistema está diseñado adecuadamente y puede competir eficazmente en la detección de piel en imágenes. Ahora debemos comprobarlo con los conjuntos con los que no ha sido entrenado.

# Validation Dataset

## 1. Sistema propio:

Si aplicamos nuestro propio sistema, los resultados son los siguientes:



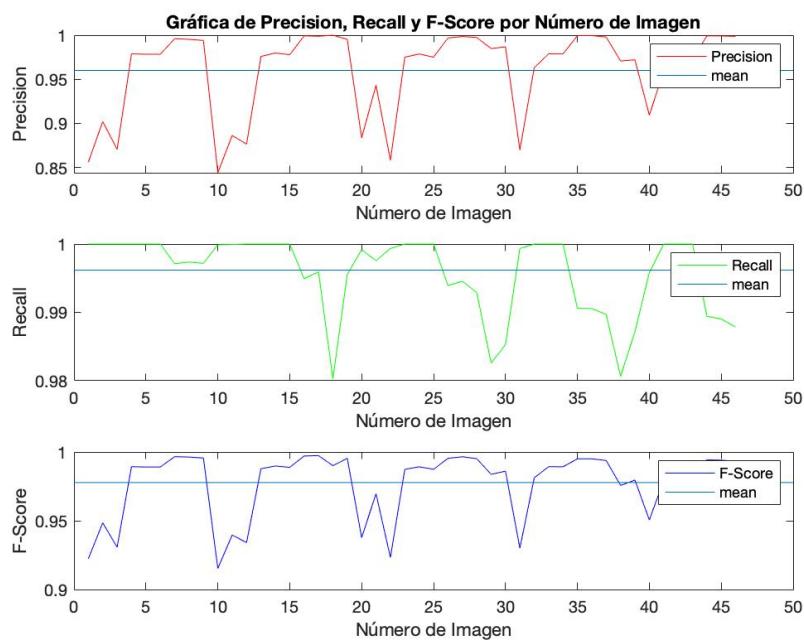
Al observar detenidamente el conjunto proporcionado, se observa que las imágenes de validación carecen de sombras. En consecuencia, nuestra funcionalidad de reducción de sombras no solo resulta innecesaria, sino que también podría introducir una ligera ralentización en el proceso.



Cuando desactivamos la opción de reducción de sombras, observamos que el valor del F-Score experimenta una leve mejora, pasando de 0.97728 a 0.97777. Esta mejora se debe al hecho de que, en un conjunto de datos sin sombras, el sistema de reducción de sombras resulta superfluo y podría, en casos excepcionales, eliminar pequeñas cantidades de píxeles de piel de manera innecesaria.

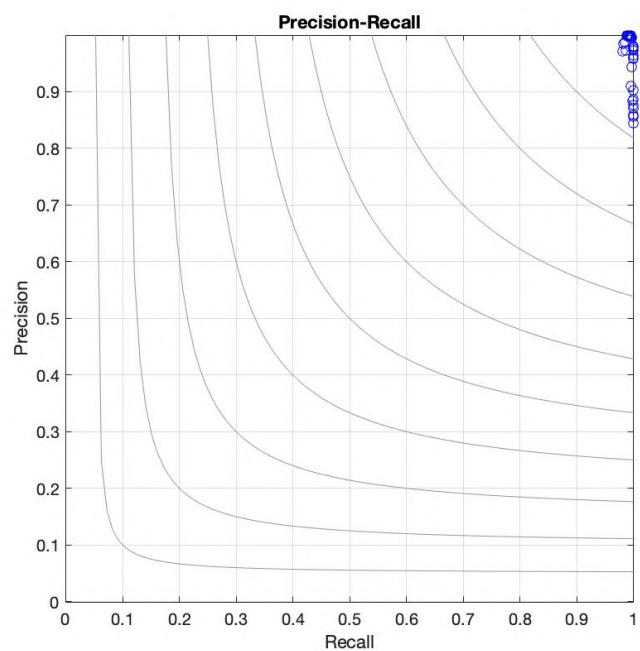
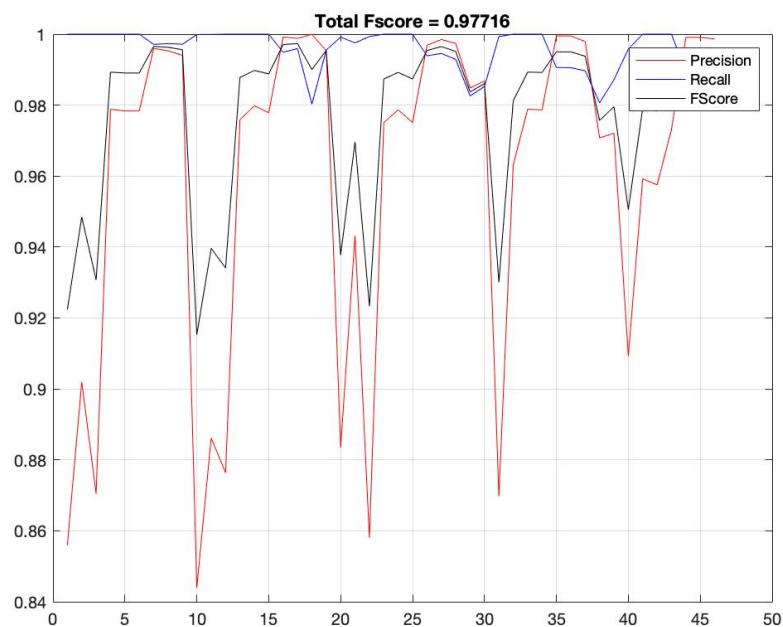
Es precisamente para afrontar este tipo de situaciones, por lo que hemos diseñado la interfaz de nuestro sistema de manera que un usuario informado pueda tomar decisiones acerca de qué sistemas adicionales de postprocesamiento activar o desactivar. No obstante, como diseñadores del sistema, recomendamos la activación de todos los sistemas en la mayoría de los casos, salvo situaciones excepcionales donde se requieran pruebas específicas o el usuario desee refinar las máscaras en contextos concretos.

Las gráficas con las métricas obtenidas:



## 2. Sistema profesorado:

Si aplicamos la herramienta de comprobación proporcionada por el profesorado, obtenemos:



### **3. Conclusiones:**

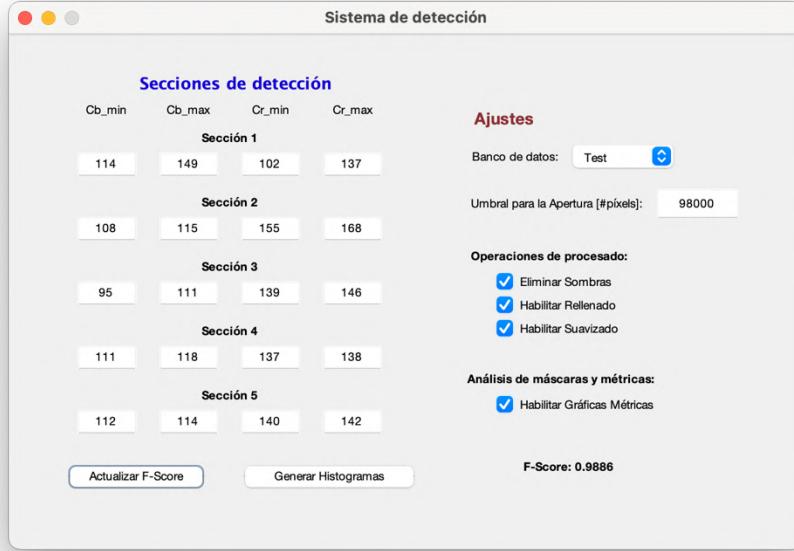
Aunque se ha observado una ligera disminución en el rendimiento en comparación con el conjunto de entrenamiento, este cambio ha sido muy pequeño. Obteniendo en la validación un 0.977 aproximadamente. Esto sugiere que nuestro sistema es altamente robusto.

El valor de precisión es ligeramente inferior, lo que afecta al F-Score. No obstante, la exhaustividad (recall) es prácticamente perfecta, lo que significa que el sistema no pasa por alto píxeles de piel, pero podría etiquetar algunos píxeles adicionales como piel de forma errónea. Estos datos indican que el sistema todavía es altamente competente y preciso en un escenario diferente al de entrenamiento, lo que resalta su flexibilidad y adaptabilidad.

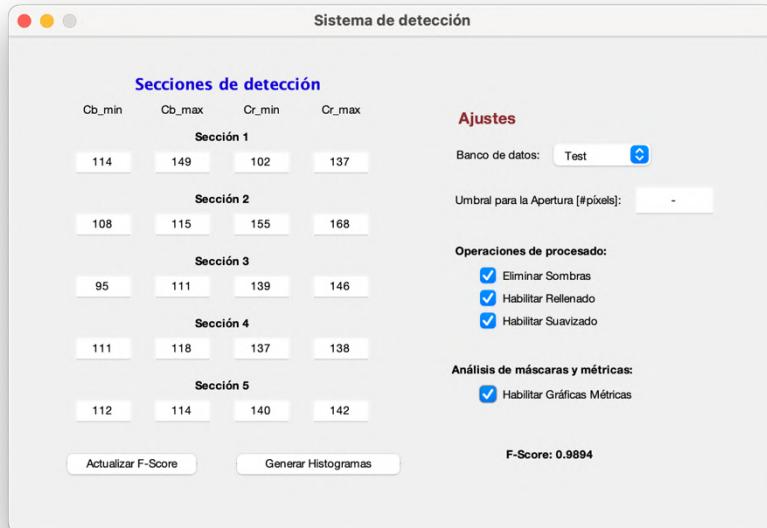
# Test Dataset

## 1. Sistema propio:

Si aplicamos nuestro propio sistema, los resultados son los siguientes:



En un caso similar al caso anterior, observamos que no existen elementos externos de gran tamaño que puedan generar confusiones notables en nuestro sistema, por lo que podemos desactivar sin problema el proceso de umbral de apertura.

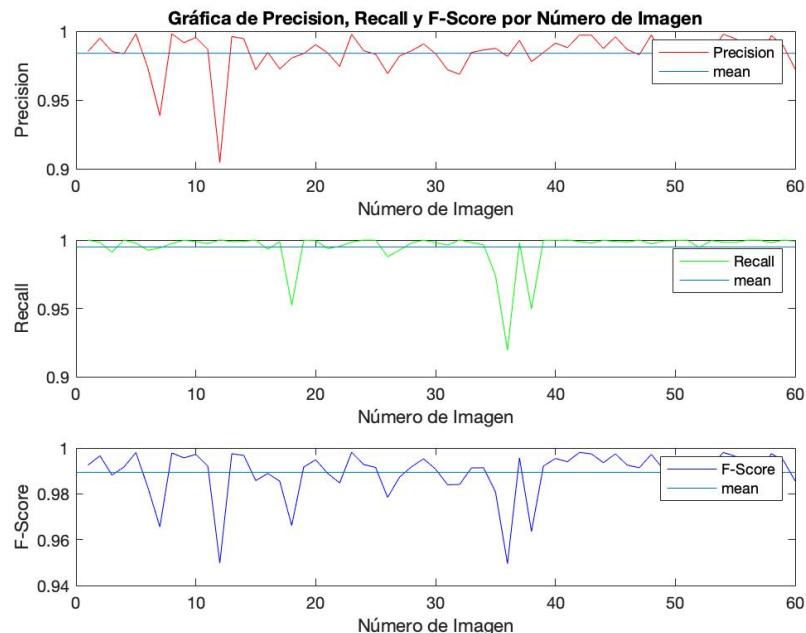


A diferencia de lo que ocurre en otros conjuntos de datos como, por ejemplo, el de entrenamiento, donde muchas imágenes presentaban elementos con tonalidades cromáticas similares a la piel generando así errores de detección, nuestro conjunto no presenta ninguna evidencia de ellos.

Es por eso por lo que en este caso podemos desactivar el umbral de apertura que facilita la eliminación de elementos interferentes de gran tamaño sin ningún problema. Al hacerlo, obtenemos una ligera mejor del F-Score de 0.9886 a 0.9894 pues al final cualquier operación morfológica innecesaria puede perjudicar levemente a las máscaras.

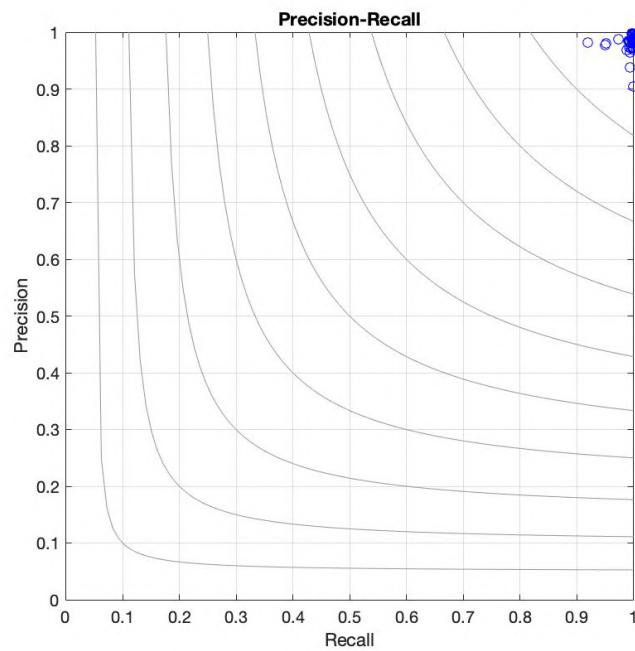
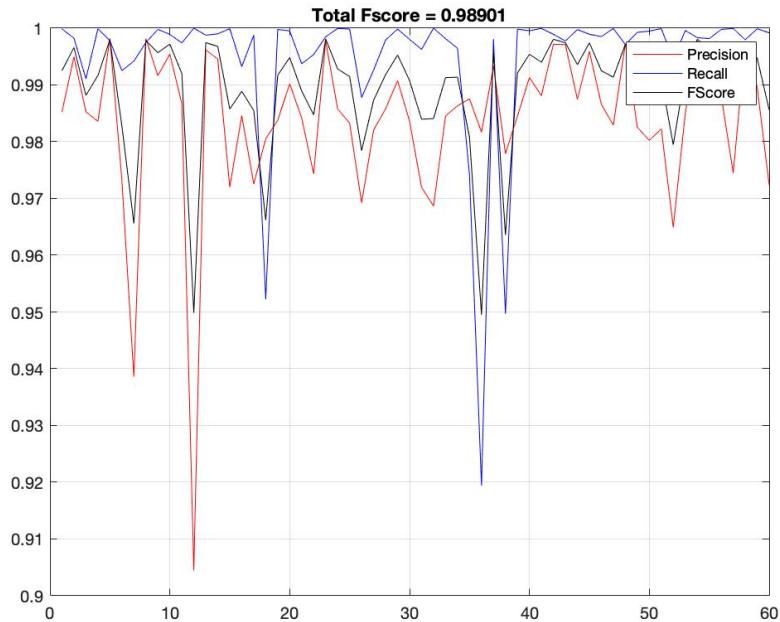
No obstante, es importante destacar que las mejoras propuestas están demostrando ser muy eficientes en la gran mayoría de casos y que en los casos muy específicos donde esta decisión puede tener un efecto perjudicial, dicho efecto resulta insignificante. En su lugar, se considera más relevante tener en cuenta este aspecto desde la perspectiva de la velocidad de procesamiento, que sería la razón principal por la que desactivaríamos procesos inútiles, para ahorrar tiempo de procesado.

Las gráficas con las métricas obtenidas:



## 2. Sistema profesorado:

Si aplicamos la herramienta de comprobación proporcionada por el profesorado, obtenemos:



### **3. Conclusiones:**

Al analizar los resultados encontramos que nuestro sistema ha obtenido un F-Score de 0.9894 con nuestros análisis y uno de F-Score de 0.98901 con la herramienta del profesorado.

Esta diferencia entre ambos resultados sigue siendo mínima lo cual no es problema ya que seguramente se debe a errores de truncamiento o redondeo en los cálculos, como ya hemos comentado. La variación es tan pequeña que no afecta significativamente la evaluación del rendimiento. Ambos valores de F-Score son altamente satisfactorios y reflejan una detección de piel muy precisa y eficiente en el conjunto de pruebas.

En este último conjunto, logramos obtener la métrica F-Score más alta entre las tres evaluadas, casi alcanzando el valor perfecto de 1. Esto es un logro significativo, ya que inicialmente no teníamos conocimiento alguno sobre este conjunto de datos y nuestro sistema no fue entrenado con él.

Haciendo una media de todos los F-Score obtenemos:

- **Evaluando con el sistema propio:**

$$\overline{F - Score_{total}} = \frac{0.98048 + 0.97777 + 0.9894}{3} = \mathbf{0.98255}$$

- **Evaluando con el sistema del profesorado:**

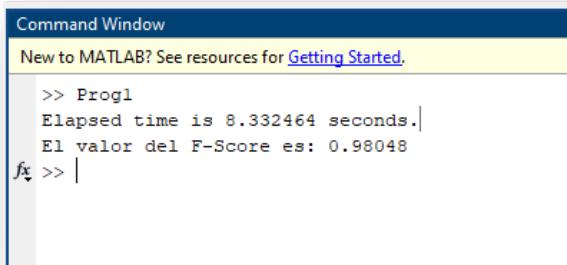
$$\overline{F - Score_{total}} = \frac{0.9804 + 0.97716 + 0.98901}{3} = \mathbf{0.98219}$$

La destacada precisión y eficiencia de nuestro sistema son claramente visibles, y la coherencia de su rendimiento en distintos conjuntos de datos refuerza su confiabilidad y versatilidad.

- **Tiempo de cálculo<sup>1\*</sup> en obtener resultados para una imagen (criterio 2)**

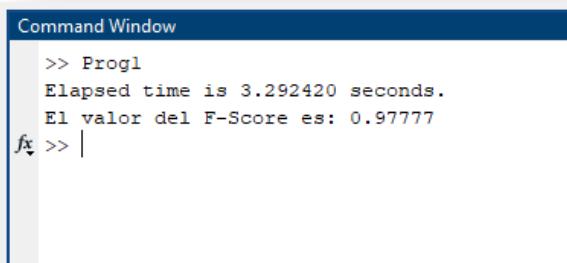
Se ha evaluado el tiempo necesario para procesar las imágenes en cada uno de los conjuntos de datos basado en el tiempo de procesado de los ordenadores del laboratorio. Los resultados se resumen a continuación:

- **Conjunto de Entrenamiento (Training):**
  - o 8.33 segundos/60 imágenes



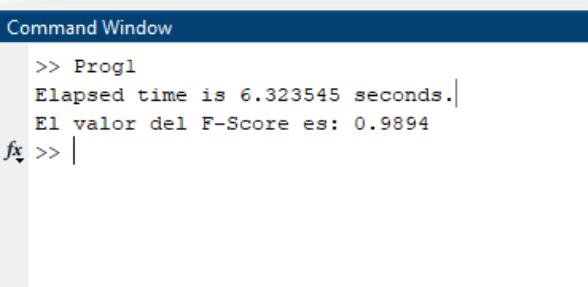
A screenshot of a MATLAB Command Window titled "Command Window". It displays the following text:  
>> Progl  
Elapsed time is 8.332464 seconds.  
El valor del F-Score es: 0.98048  
fx >> |

- **Conjunto de Validación (“Validation”):**
  - o 3.29 segundos/60 imágenes



A screenshot of a MATLAB Command Window titled "Command Window". It displays the following text:  
>> Progl  
Elapsed time is 3.292420 seconds.  
El valor del F-Score es: 0.97777  
fx >> |

- **Conjunto de Pruebas (“Test”):**
  - o 6.32 segundos/60 imágenes



A screenshot of a MATLAB Command Window titled "Command Window". It displays the following text:  
>> Progl  
Elapsed time is 6.323545 seconds.  
El valor del F-Score es: 0.9894  
fx >> |

Estos resultados se han evaluado poniendo el cronómetro temporizador de Matlab en el inicio y el final de toda la ejecución. No obstante, ese valor corresponde a las 60 imágenes analizadas, ahora debemos calcular el valor individual por imagen:

- **Conjunto de Entrenamiento (Training):**
  - o 138.87ms/imagen
  
- **Conjunto de Validación (“Validation”):**
  - o 54.87ms/imagen
  
- **Conjunto de Pruebas (“Test”):**
  - o 105.32ms/imagen

Al realizar un cálculo promedio de los 3 conjuntos, se obtiene un tiempo promedio por imagen de:

- **Promedio Total:**

$$\bar{t} = \frac{(6.323545 + 3.29242 + 8.332464)^s / 60 \text{ imágenes}}{3} \cdot \frac{1000ms}{s}$$

$$= \mathbf{99.71 \text{ ms/imagen}}$$

Estos valores representan el tiempo requerido para ejecutar todo el proceso de detección de piel en una sola imagen, lo que proporciona una idea clara del rendimiento y la eficiencia del sistema en términos de velocidad de procesamiento.

Esto significa que, en promedio, nuestro sistema es capaz de realizar la detección de piel en una imagen en un tiempo muy razonable y eficiente.

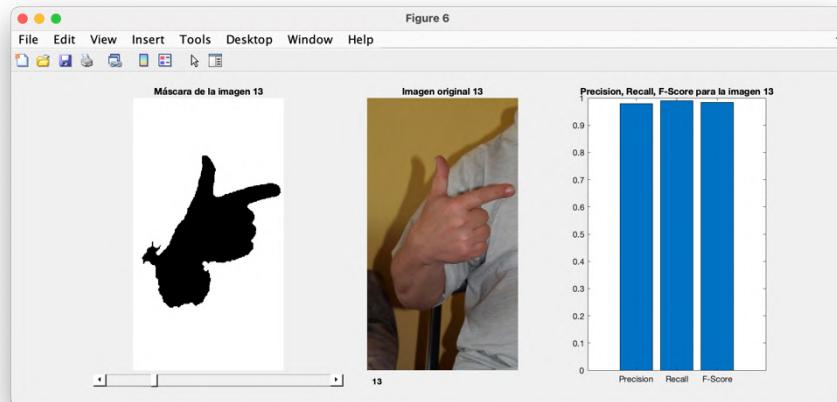
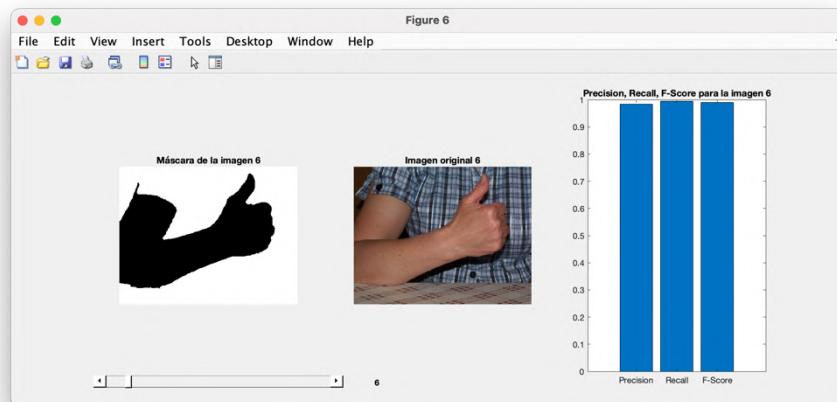
Este resultado es bastante prometedor, ya que indica que el sistema es capaz de manejar una carga de trabajo significativa, lo que lo hace adecuado para aplicaciones en tiempo real o para procesar grandes conjuntos de imágenes de manera rápida y eficaz.

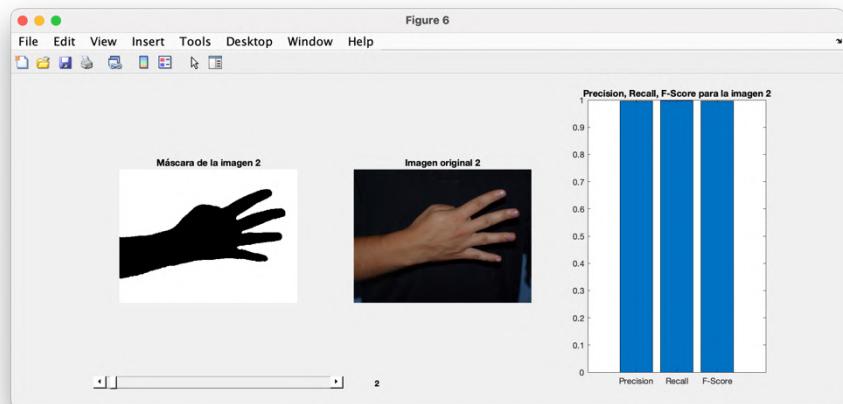
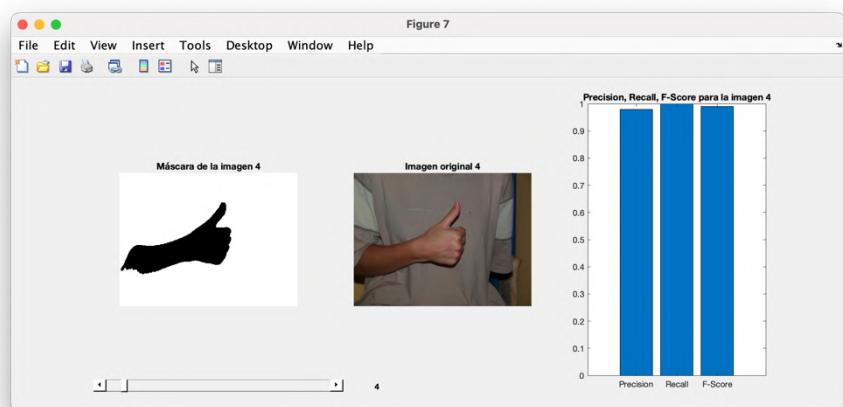
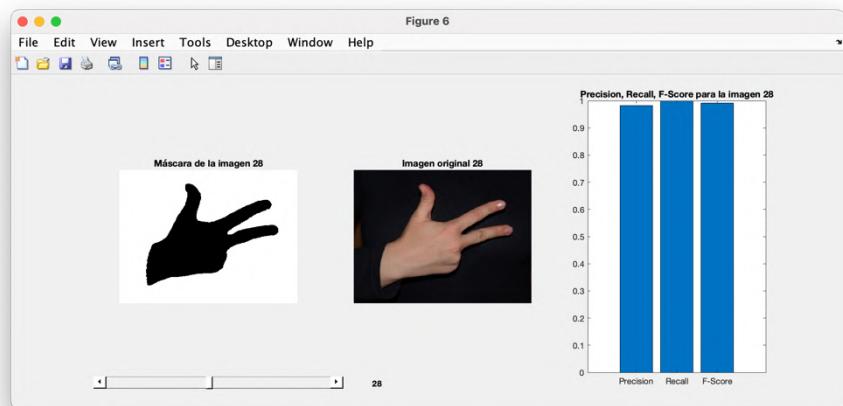
## d. Análisis de los resultados (incluyendo ejemplos de buena detección y de errores)

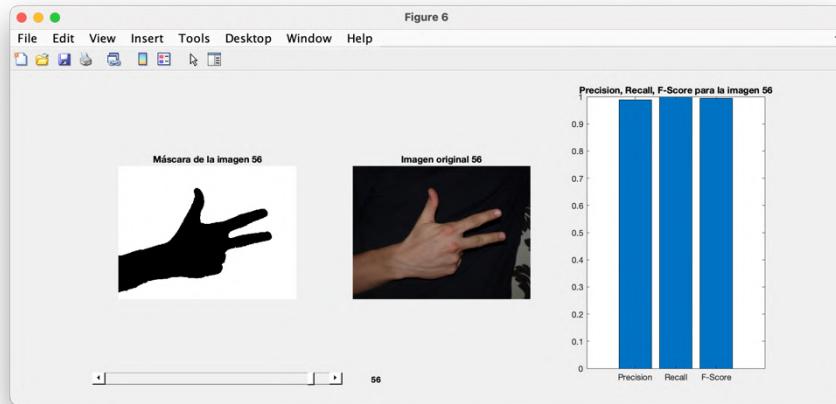
Estudio de la sensibilidad de los parámetros sobre los resultados.

Nuestro sistema de detección de piel ha proporcionado resultados consistentemente sólidos en los tres conjuntos de datos: entrenamiento, validación y prueba. A continuación, analizaremos los resultados y presentaremos ejemplos tanto de buenas detecciones como de errores notables.

**Buenas Detecciones:** En general, nuestro sistema ha demostrado ser muy eficaz en detectar regiones de piel en todas las imágenes proporcionadas. En la gran mayoría de imágenes en las que se realizó se logra un equilibrio casi perfecto entre precisión y exhaustividad de valores, además, muy altos. Esto se traduce en un alto valor de F-Score, indicando una detección precisa y exhaustiva de píxeles de piel. A continuación, se muestran diversos ejemplos de buena detección en una imagen del todos los conjuntos:





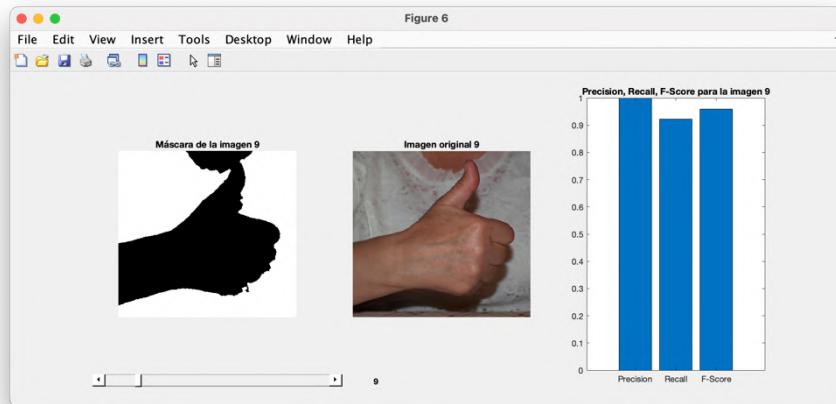


Observamos si vamos una a una por las 180 imágenes que en la gran mayoría de ellas las manos presentan una máscara casi perfecta. Sin embargo, existen algunos casos muy concretos donde esto no ocurre y los comentaremos a continuación.

**Errores Notables:** A pesar de la eficacia general de nuestro sistema, existen situaciones muy concretas en las que se pueden producir errores relativamente significativos. Estos errores suelen ocurrir cuando ocurre alguna de las siguientes situaciones:

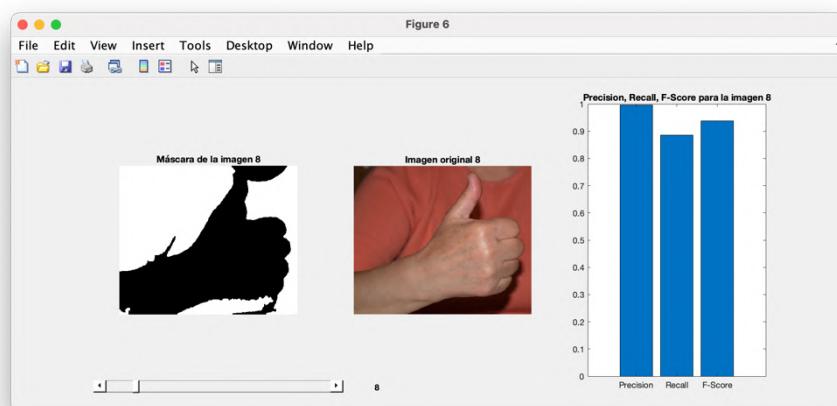
i. **Partes del cuerpo en contacto con la propia mano:**

- a. En casos donde las manos presentan regiones de piel en contacto directo con otras zonas de piel, como cuellos por ejemplo, nuestro sistema puede tener dificultades para separar estas áreas con precisión.

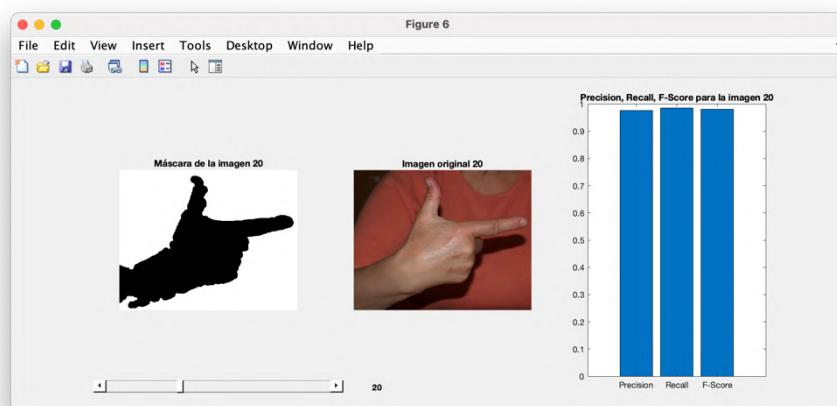


ii. Sombras sobre colores de tonos cromáticos similares a la piel:

- a. Nuestro sistema de reducción de sombras es efectivo en la gran mayoría de los casos, pero como ya desveló nuestro informe hay determinados tonos de sombras que comparten tonalidad cromática con la piel. Puede haber situaciones donde las sombras, especialmente si se superponen con áreas de tonos cromáticos similares a la piel, se generen unos pocos falsos positivos.

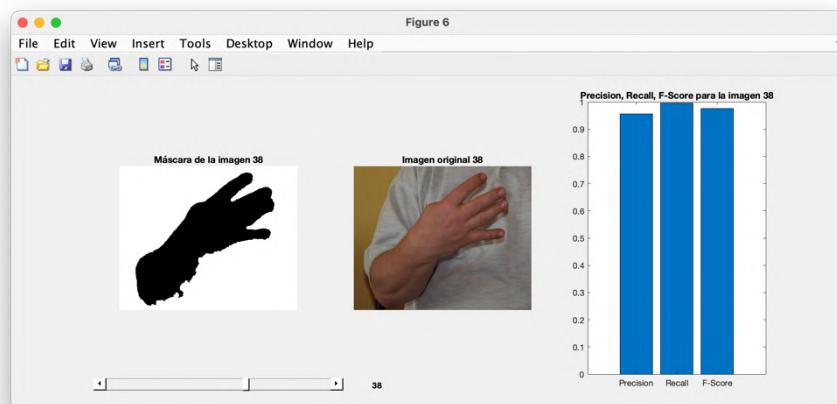


- b. En la gran mayoría de casos nuestro sistema si logra eliminar estas sombras de falsos positivos perjudicando muy ligeramente el contorno de la mano en el proceso.



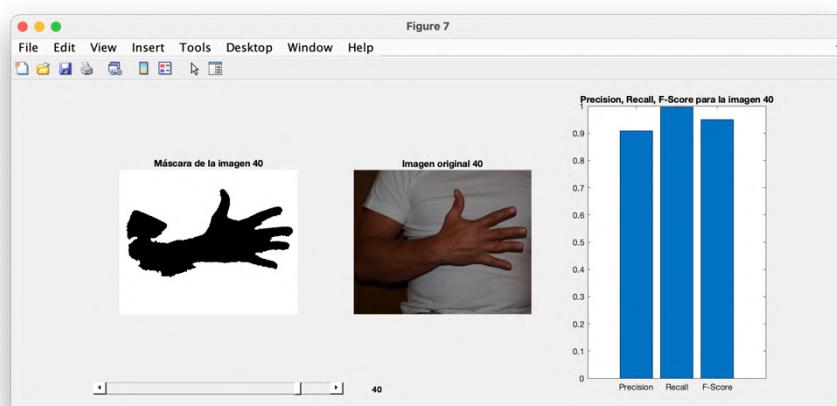
**iii. Dedos excesivamente juntos:**

- a. Cuando los dedos están demasiado juntos o incluso en contacto, nuestro sistema puede tener dificultades para separarlos y detectar cada dedo con precisión, lo que a posteriori el siguiente sistema a diseñar podría confundirlos por un único dedo.



**iv. Pérdida de detalle en zonas oscuras:**

- a. En ocasiones, en zonas muy oscuras de las imágenes, nuestro sistema podría tener dificultades para detectar con precisión el contorno de los píxeles de piel debido a la pérdida de detalles en estas áreas, pero sí logaría detectar con bastante precisión la mayor parte de la piel.



A pesar de estos errores, es importante destacar que representan situaciones excepcionales, y por tanto un porcentaje poco significativo del conjunto. Nuestro sistema logra una detección precisa y exhaustiva de la piel. La robustez y eficacia general del sistema son notables, y estos casos excepcionales no afectan significativamente a su rendimiento en la detección de piel en imágenes.

### **Mejoras Implementadas:**

Durante el desarrollo de nuestro sistema, hemos introducido diversas mejoras para optimizar su rendimiento. Estas mejoras incluyen la implementación de procesos de reducción de sombras, eliminación de elementos interferentes, rellenado de máscaras y suavizado de bordes. Cada uno de estos procesos ha demostrado ser esencial en diferentes contextos, contribuyendo a mejorar tanto la precisión como la exhaustividad de la detección de piel, como ya se ha demostrado anteriormente. Razón por la cual no repetiremos aquí la importancia ni los ejemplos pues ya ha sido analizada al detalle anteriormente.

No obstante, si comentaremos que parámetros puede modificar la sensibilidad y el refinado de las máscaras.

### **Estudio de la Sensibilidad de los Parámetros:**

Nuestro sistema de detección de piel cuenta con varios parámetros que pueden afectar el rendimiento. A lo largo del desarrollo, hemos evaluado cuidadosamente la sensibilidad de estos parámetros para definirlos de forma óptima en términos del valor F-Score. Algunos de los parámetros que hemos explorado incluyen:

1. **Regiones de detección:** La elección de unas regiones de detección diferentes afectaría significativamente a la detección. Nosotros hemos definido el número de regiones y los márgenes de estas a partir de algoritmos que nos maximizaran la métrica F-Score, por tanto, las óptimas de forma empírica. El uso de otras regiones afectaría directamente a que zonas se definen como regiones de detección de piel.
2. **Umbral de Apertura:** Hemos observado que la elección del umbral de apertura puede tener un impacto significativo en la detección de piel. El umbral de apertura es el valor a partir las imágenes con máscaras de mayor tamaño que ese umbral sufrirán un proceso de erosión y dilatación para ayudar a la eliminación de elementos interferentes. Esto en la mayoría de los casos era un proceso perjudicial porque perdíamos detalle en las máscaras que

ya eran casi ideales. No obstante, en los casos que la máscara era demasiado grande (razón por la que intuíamos que había elementos interferentes formando parte) este proceso era de gran utilidad. Por ellos la elección de valores bajos pueden conducir a una mayor precisión, pero a expensas de la exhaustividad, mientras que valores altos pueden aumentar la exhaustividad, pero disminuir la precisión. Debe encontrarse un equilibrio para optimizar la utilidad de este proceso.

3. **Procesos de Postprocesamiento:** La habilitación o deshabilitación de los procesos de reducción de sombras, relleno y suavizado de bordes ha demostrado ser crucial. Cada uno de estos procesos puede tener un impacto en el F-Score, y su sensibilidad varía según el conjunto de datos. Generalmente la habilitación de todos ellos es la mejor opción, pero en casos muy concretos deshabilitar alguna de las 3 opciones puede mejorar muy ligeramente el F-Score y reducir el tiempo de ejecución.
4. **Elemento Estructural:** La elección del elemento estructural utilizado en las operaciones morfológicas, como son la apertura comentada en el punto 2 para eliminar elementos interferentes, como a la hora de suavizar los bordes de la máscara, es de crucial importancia. Hemos evaluado cuidadosamente todos los elementos estructurales típicos, e incluso algunas opciones atípicas como la conversión a 2D de elementos 3D, para determinar cuáles eran los óptimos para su correspondiente propósito. La elección de otro elemento afectaría directamente a los resultados obtenidos.
5. **Conjunto de datos:** Aunque de primeras parece obvio, cabe decir que tanto el conjunto de datos utilizado para entrenar como el conjunto donde se evalúe son claves en la obtención de resultados, una mayor densidad de información para entrenar resultará en un sistema más robusto, no obstante, la complejidad y la calidad de las imágenes que se analizan siempre afectará a las métricas y los resultados.

El estudio de la sensibilidad de estos parámetros nos ha permitido ajustar nuestro sistema para obtener los mejores resultados en una amplia variedad de situaciones. En general, hemos encontrado que la versatilidad y la capacidad de adaptación de nuestro sistema son sus puntos fuertes, lo que le permite funcionar bien en diferentes contextos y con diversos tipos de imágenes.

## Resultados de Tiempo de Ejecución:

El tiempo de cálculo de nuestro sistema se ha medido en tres conjuntos de datos diferentes: entrenamiento, validación y prueba. Los tiempos de cálculo fueron de 8.33 segundos, 3.29 segundos y 6.32 segundos, respectivamente, para todo el conjunto de imágenes. Esto se traduce en un tiempo promedio de procesamiento por imagen de aproximadamente 99.71 ms. Podemos destacar la eficiencia en términos de tiempo de procesamiento como uno de los aspectos sobresalientes de nuestro sistema, lo que lo convierte en una herramienta viable para aplicaciones prácticas en las que la velocidad de procesamiento es esencial.

## Resultados de F-Score:

Los valores de la métrica F-Score arrojan resultados altamente positivos, con una F-Score promedio de 0.98255 para el conjunto completo. Estos resultados confirman la robustez y eficiencia de nuestro sistema.

## Interfaz Accesible:

La interfaz de nuestro sistema ha sido diseñada para ser accesible y fácil de usar. Permite a los usuarios tomar decisiones informadas sobre qué procesos de postprocesamiento activar o desactivar, qué parámetros desean editar y observar al detalle cómo afecta a sus resultados, a través de diferentes mecanismos, lo que proporciona flexibilidad y control. Aunque recomendamos la activación de todos los sistemas en la mayoría de los casos, esta capacidad de personalización permite a los usuarios adaptar el sistema a sus necesidades específicas.



**e. Finalmente, presentación de ideas para mejorar el sistema.**

**1. Conjunto de Entrenamiento Mayor:**

- a. Una forma efectiva de mejorar el sistema es ampliar el conjunto de entrenamiento. Cuanto más variado y extenso sea el conjunto de imágenes utilizadas para entrenar el modelo, mejor será su capacidad para generalizar y detectar la piel en diversas condiciones.
- b. Un conjunto de imágenes mayor nos permitiría desde un inicio definir más de forma más detallada, y para un conjunto mucho mayor, las regiones de detección de piel, y poder evaluar los resultados con un mayor número de muestras. Esto nos permitiría evaluar resultados basados en un mayor número de muestras y, como resultado, definir posibles mejoras o algoritmos que faciliten la detección o mejoren el procesamiento de las máscaras.
- c. En un banco de datos reducido, existe la incertidumbre de si el sistema implementado es exclusivamente efectivo en las imágenes con las que ha sido entrenado, o si su eficacia se extiende a todas las posibles imágenes. La limitación de tamaño de la muestra impide una evaluación exhaustiva.
- d. En nuestro sistema actual, hemos desarrollado mejoras de manera cuidadosa, demostrando ser altamente beneficiosas en la mayoría de las situaciones. Sin embargo, un conjunto de entrenamiento mucho más extenso nos brindaría la libertad de desarrollar algoritmos más sofisticados sin el riesgo de adaptarnos únicamente a los detalles de nuestras imágenes de muestra, permitiendo una detección de piel más generalizada y precisa.

## 2. Regiones Elípticas:

- a. Durante el análisis del histograma de crominancias CbCr, se observó que la distribución de los píxeles de piel no se asemejaba a una región cuadrada, sino más bien presentaba patrones ligeramente elípticos. Esta característica impedía la definición de una región de detección cuadrada que incluyera todos los píxeles de piel sin incluir otros tipos de píxeles. Para abordar este desafío, aplicamos un enfoque basado en definir más cantidad de regiones rectangulares, pero de menor tamaño, basándonos en la optimización del F-Score. Este enfoque nos permitió determinar el número óptimo de regiones rectangulares y sus márgenes para la detección de piel. Empíricamente, hemos demostrado que las regiones rectangulares generan resultados altamente efectivos, y que las nuestras en concreto son las óptimas para este enfoque, como se refleja en las métricas de F-Score y las máscaras generadas.
- b. A pesar de la eficacia de las regiones rectangulares, existe una posibilidad de implementar regiones elípticas, lo que podría adaptarse de manera más precisa a la forma natural de la distribución de píxeles en el histograma. Un análisis detallado sería necesario para optimizar la cantidad y los márgenes de estas regiones elípticas, siguiendo un proceso similar al que aplicamos para las regiones cuadradas. Esta adaptación podría conducir a una mejora adicional en la precisión de la detección.
- c. Si bien eso podría ayudar, debemos destacar que ya hemos considerado algoritmos que abordan eficazmente la problemática implícita del uso de regiones rectangulares. El principal de ellos es la reducción de sombras, que fue diseñada a partir de un banco de datos exclusivamente compuesto por sombras, provenientes del conjunto de entrenamiento original. Este algoritmo permite eliminar los píxeles o pequeñas regiones correspondientes a sombras o elementos interferentes que se encuentran dentro de las propias regiones de detección previamente definidas. Como resultado, las regiones de piel se refinan aún más al eliminar los píxeles que no corresponden a las zonas de piel. Esta mejora es un ejemplo de cómo hemos optimizado nuestras regiones cuadradas para lograr una detección precisa de la piel, y como a partir de ciertas medidas hemos logrado optimizar nuestro enfoque a este desafío.

### 3. Distinción de Partes del Cuerpo:

- a. Suponiendo que el sistema se utilice exclusivamente para la detección de manos, sería beneficioso optimizar aún más su rendimiento al permitir que distinga entre áreas de piel que corresponden a manos y áreas de piel de otras partes del cuerpo. Esto podría lograrse mediante un proceso de segmentación que se centre en las siguientes etapas:

#### i. Fase 1: Segmentación de Zonas de Interés:

- **Detección de Características Específicas:** El algoritmo comenzaría identificando características específicas que distinguen las manos de otras partes del cuerpo, como la forma, el tamaño, la ubicación y las características de la piel. Esto podría lograrse mediante técnicas de procesamiento de imágenes y visión por computadora, como la detección de bordes, la detección de regiones, la extracción de características y la segmentación.
- **Identificación de la Región de las Manos:** Una vez detectadas las características de las manos, el algoritmo identificaría la región que contiene las manos en la imagen. Esto podría lograrse utilizando algoritmos de detección de objetos o técnicas de aprendizaje automático, como el uso de clasificadores entrenados previamente para reconocer las manos.

#### ii. Fase 2: Refinamiento de Regiones de Piel:

- **Definición de Bordes de la Mano:** Con la región de las manos identificada, el algoritmo se centraría en refinar aún más las áreas de piel que corresponden a las manos. Se utilizarían técnicas de detección de bordes y segmentación para definir con precisión los límites de las manos y separarlas de otras partes del cuerpo.
- **Eliminación de Ruidos e Interferencias:** Para mejorar la precisión de la segmentación de las manos, el algoritmo aplicaría técnicas de eliminación de ruidos, como la propia reducción de sombras, la eliminación de objetos no deseados a partir de los algoritmos ya definidos y de otros nuevos. Esto garantizaría que solo se retengan las regiones de piel relacionadas con las manos.

iii. **Fase 3: Comparación y Validación:**

- **Comparación con Patrones de Mano:** Las regiones de piel refinadas se compararían con patrones conocidos de manos humanas extraídas de la información de máscaras ideales del conjunto de entrenamiento. Estos patrones serían modelos previamente entrenados que representan la forma y las características de las manos. La comparación se basaría en métricas como la similitud de forma y textura.
- **Validación de Regiones de Mano:** Las regiones que cumplan con los criterios establecidos, como una alta similitud con los patrones de manos y la presencia de características distintivas de estas, se validarían como regiones de mano. Aquellas que no cumplen con estos criterios se descartarían.

iv. **Fase 4: Generación de Máscaras de Mano:**

- **Generación de Máscaras de Mano:** Finalmente, se generaría una máscara de mano basada en las regiones de piel identificadas como manos. Esta máscara se utilizaría para representar de manera binaria el área de interés de la imagen.

#### 4. Detección en Tiempo Real:

- a. La capacidad de optimizar nuestro sistema de detección de piel para operar en tiempo real sería un paso fundamental para su aplicación en una amplia gama de escenarios prácticos. A continuación, se detalla cómo podríamos abordar esta mejora:

##### i. Fase 1: Optimización del rendimiento:

- o **Procesamiento Paralelo:** Una de las estrategias clave para lograr la detección en tiempo real es aprovechar al máximo la capacidad de procesamiento de las unidades de procesamiento gráfico (GPU) y las unidades de procesamiento central (CPU). Implementar algoritmos de procesamiento paralelo permitiría realizar múltiples tareas simultáneamente, acelerando significativamente la velocidad de procesamiento.
- o **Reducción de Carga Computacional:** Identificar y eliminar procesos o funciones que no sean esenciales para la detección de piel en tiempo real. Esto podría incluir la reducción de operaciones innecesarias o redundantes, así como la optimización de algoritmos y estructuras de datos.

##### ii. Fase 2: Implementación de Detección en Tiempo Real:

- o **Interfaz de Adquisición de Imágenes en Tiempo Real:** El sistema requeriría una interfaz de adquisición de imágenes en tiempo real que permita la captura continua de imágenes desde una cámara o fuente de video en directo. Esto implicaría la integración de bibliotecas y tecnologías de adquisición de video en la aplicación.
- o **Segmentación Eficiente de Imágenes en Tiempo Real:** Se deben aplicar algoritmos de segmentación de imágenes de alto rendimiento que sean capaces de procesar imágenes en tiempo real sin demoras perceptibles. La elección de algoritmos y parámetros de segmentación eficientes sería esencial para lograr este objetivo.

##### iii. Fase 3: Pruebas y Ajustes:

- o **Pruebas en Tiempo Real:** Una vez que se haya implementado la detección en tiempo real, es fundamental realizar pruebas exhaustivas en escenarios de uso real. Esto

implicaría el análisis de la eficacia, precisión y velocidad del sistema en situaciones prácticas.

- **Ajustes en Tiempo Real:** Durante las pruebas, se identificarían posibles desafíos o problemas que deberían ajustarse al mismo tiempo que se ejecuta la aplicación.

## **5. Mejora de la Interfaz de Usuario:**

- a. El desarrollo de una interfaz de usuario más intuitiva y accesible para usuarios no técnicos sería un paso importante para aumentar la usabilidad de nuestro sistema de detección de piel. Esta mejora puede incluir varios elementos clave para hacer que el sistema sea más fácil de comprender y utilizar:
  - i. **Manual de Usuario Detallado:** Un manual de usuario completo y fácil de entender sería fundamental para que los usuarios no técnicos comprendan el funcionamiento del sistema. Este manual proporcionaría una descripción detallada de los diferentes parámetros, sus funciones y su influencia en los resultados de detección de piel. Además, explicaría de manera clara y concisa cómo interpretar las métricas y los resultados obtenidos, como el F-Score y otros indicadores relevantes.
  - ii. **Asistencia para la Interpretación de Resultados:** Junto con el manual de usuario, la interfaz de usuario podría incluir funciones de ayuda o información contextual que expliquen conceptos clave relacionados con la detección de piel además de ejemplos que muestren las posibilidades del sistema.

## 6. Implementación de Redes Neuronales:

- a. La implementación de técnicas de aprendizaje profundo, como las redes neuronales convolucionales (CNN), podría ser un paso importante para mejorar aún más la precisión de la detección de piel. Las CNN son capaces de aprender automáticamente características y patrones complejos en los datos, lo que podría hacer que la detección sea más robusta y eficiente.
- b. **Beneficios de las Redes Neuronales Convolucionales (CNN):**
  - i. Extracción Automática de Características: Las CNN son capaces de aprender y extraer características relevantes de las imágenes, lo que significa que pueden identificar patrones de color, textura y forma específicos asociados con la piel humana sin una definición manual de regiones de interés.
  - ii. Adaptabilidad: Las CNN pueden adaptarse a una variedad de condiciones de iluminación, tonos de piel y contextos, lo que las hace efectivas en la detección de piel en diferentes situaciones.
  - iii. Detección de Características Multiescala: Estas redes adoptan un enfoque en procesamiento de imágenes y visión por computadora que busca detectar características, como bordes, esquinas, regiones de interés o patrones, en una imagen a diferentes escalas. Esto significa que se analiza la imagen en busca de estas características en múltiples niveles de detalle o resolución.
- c. **Posibles Pasos para la Implementación:**
  - i. Recopilación de Datos: Se requeriría un conjunto de datos grande y diverso que contenga imágenes de piel con anotaciones precisas para el entrenamiento de la red.
  - ii. Entrenamiento de la CNN: Se entrenaría una CNN utilizando este conjunto de datos, lo que permitiría a la red aprender a identificar regiones de piel en imágenes de manera efectiva.
  - iii. Validación y Ajuste Fino: La red entrenada se validaría en conjuntos de datos de prueba para

evaluar su precisión y realizar ajustes según sea necesario.

- iv. Implementación en el Sistema: Una vez que la CNN se haya entrenado y validado con éxito, se podría integrar en el sistema de detección de piel existente, mejorando la precisión y adaptabilidad del sistema.
  - v. Evaluación Continua: La CNN requeriría evaluación y ajustes periódicos a medida que se enfrenta a nuevos desafíos y variaciones en las condiciones de las imágenes.
- d. La implementación de CNN podría ser particularmente beneficiosa para mejorar la detección en situaciones difíciles. Además, permitiría una mayor generalización y adaptabilidad del sistema a diferentes aplicaciones, como la detección médica, la seguridad y las interfaces de usuario basadas en gestos.
- e. No obstante, la aplicación de este tipo de sistemas requeriría de un conjunto de datos grande y diverso con anotaciones precisas que contenga una variedad de imágenes de piel en diferentes contextos y condiciones, una cantidad significativa de recursos computacionales, conocimientos en aprendizaje profundo y visión por computadora y multitud de conjuntos de datos de prueba. Elementos y factores de los cuales, por el momento, no disponemos.

## 7. Optimización del Rendimiento:

- a. En el contexto de un sistema de detección de piel, la optimización de rendimiento nos implica realizar mejoras para reducir el tiempo necesario para procesar una imagen y disminuir la carga computacional, lo que haría que el sistema sea más adecuado para ser utilizado en dispositivos con recursos limitados, como dispositivos móviles o sistemas embebidos.
- b. Para llevar a cabo esta optimización, se podrían aplicar diversas estrategias, como:
  - i. **Optimización de algoritmos:** Revisar y modificar los algoritmos utilizados en el sistema para que sean más eficientes en cuanto a uso de recursos y tiempo de ejecución.
  - ii. **Paralelización:** Aprovechar la capacidad de procesadores multicore o GPUs para procesar varias partes de la imagen o varias imágenes simultáneamente, lo que aceleraría el tiempo de procesamiento.
  - iii. **Reducción de redundancias:** Identificar y eliminar pasos o cálculos redundantes en el proceso de detección que no contribuyan significativamente a la precisión.
  - iv. **Técnicas de compresión de datos:** Reducir el tamaño de los datos sin pérdida de información para disminuir los tiempos de transferencia y procesamiento.
  - v. **Carga perezosa:** Realizar cálculos solo cuando sea necesario, en lugar de procesar toda la imagen por adelantado si la aplicación concreta a la que se aplica no lo requiere.
- c. La optimización del rendimiento no solo mejora la velocidad de procesamiento, sino que también reduce el consumo de energía y asegura que el sistema sea más accesible en una variedad de plataformas y escenarios de uso. Algo que, como hemos visto, es especialmente relevante en aplicaciones en tiempo real, donde se requiere una rápida respuesta del sistema.

## 8. Aplicaciones en Detección Médica:

- a. Aunque el sistema se ha diseñado con un enfoque en la detección de manos, existe un potencial considerable para considerar su aplicación en campos médicos específicos. Por ejemplo, en áreas como la dermatología, este sistema podría ser de gran utilidad para el diagnóstico y el seguimiento de condiciones de la piel, como dermatitis, quemaduras, erupciones cutáneas, y más.
- b. El sistema podría utilizarse en aplicaciones médicas para evaluar y analizar imágenes de la piel del paciente, permitiendo una detección temprana de problemas dermatológicos o cambios en la piel que presentaran patrones cromáticos atípicos en la piel. Como nuestro sistema es capaz de detectar zonas piel con gran precisión también es capaz de detectar por el contrario la ausencia de esta. Es decir que podría detectar zonas anómalas en la piel.
- c. Algunos ejemplos de aplicaciones específicas incluyen:
  - i. **Seguimiento de Erupciones Cutáneas:** En casos de pacientes con afecciones crónicas, como psoriasis o eczema, el sistema podría registrar y analizar imágenes de las erupciones para evaluar la gravedad, identificar mejorías o detectar nuevos brotes.
  - ii. **Evaluación de Quemaduras:** En el caso de quemaduras, el sistema podría ayudar como proceso de detección preliminar a los profesionales de la salud para evaluar la profundidad y gravedad de la lesión, lo que guiaría las decisiones de tratamiento.
  - iii. **Diagnóstico de Enfermedades de la Piel:** Podría utilizarse para el diagnóstico de enfermedades de la piel, como la dermatitis atópica o la rosácea, al identificar patrones y síntomas característicos en imágenes de la piel.
  - iv. **Seguimiento de la Evolución de Úlceras y Heridas:** En pacientes con úlceras o heridas crónicas, el sistema podría evaluar la evolución y curación de estas lesiones a lo largo del tiempo y

proporcionar métricas que cuantifiquen el proceso de curación.

v. **Diagnóstico de Infecciones Cutáneas:** El sistema podría identificar posibles signos visuales de infecciones cutáneas sin necesidad de un profesional de la salud, lo que permitiría una detección temprana y un tratamiento oportuno.

d. El sistema podría ser especialmente valioso en la telemedicina y el seguimiento a distancia de afecciones dermatológicas como las que se han comentado, donde los pacientes pueden capturar imágenes de sus afecciones y evaluar su situación, compartiendo sus resultados con profesionales de la salud para su evaluación si fuera necesario. Esto mejoraría la accesibilidad a la atención médica y permitiría un seguimiento continuo de las afecciones de la piel.

## 9. Histogramas de 128 Niveles:

- En la fase final del desarrollo de este detector de píxeles de piel, surgió la pregunta de si la variación en el tamaño de los histogramas utilizados para cuantificar los valores de los píxeles en el espacio de Cb y Cr podría influir en el F-Score, la métrica cuantificable que mide la eficiencia del detector. Y aunque quizás este aspecto ha pasado desapercibido durante el desarrollo, es fundamental para el funcionamiento del detector. Todas las técnicas de enmascaramiento y detección se aplicaron en un espacio de píxeles con valores de Cb y Cr cuantificados entre 0 y 255, es decir, siempre trabajamos con valores de 8 bits. Por tanto, surgió la curiosidad de explorar cómo funcionaría el detector con histogramas de menor precisión.
- Una pregunta que podría surgir es por qué decidimos experimentar con histogramas de "menor precisión" o, en otras palabras, reducir la cantidad de bits utilizados para representar los valores Cb y Cr de los píxeles. La motivación detrás de esta experimentación radica en el concepto de "sobreajuste" (over-fitting). El sobreajuste se refiere a entrenar un modelo para que funcione muy bien con una distribución de píxeles en el conjunto de entrenamiento, pero debido a su alta resolución y ajuste a ese conjunto específico de imágenes, podría no funcionar tan bien con otro conjunto de imágenes. Esto se debe a que la distribución de píxeles de piel en el espacio de color elegido podría variar ligeramente, y el detector no sería lo suficientemente general para obtener buenos resultados de F-Score en esos conjuntos de imágenes algo distintos.
- Por otro lado, existe el concepto opuesto: reducir la cantidad de bits utilizados para construir el histograma a un nivel tan bajo que los píxeles de piel y no piel se confundan por completo. Cuando dos píxeles (uno de piel y otro no) se agrupan en el mismo bin del histograma, no es posible distinguirlos. Esta reducción de la cantidad de bits utilizados para la cuantificación de los valores de Cb y Cr en un histograma a un nivel inferior (reducción de bits) se considera una operación no reversible.
- El objetivo detrás de esta exploración era encontrar la cantidad óptima de cuantificación (ya sea con 8, 7, 6, 5 bits, u otros) que, combinada con las prácticas de detección previamente establecidas en este informe, permitiera

obtener un detector con las mejores características objetivas.

#### - **Diseño del Detector Cuantificado**

- i. Para implementar esta versión cuantificada del detector, es fundamental comprender que su funcionamiento es análogo al del detector previamente demostrado. En otras palabras, los píxeles se encuentran en un espacio de color específico, en este caso, YCbCr, el mismo espacio de color que hemos estado utilizando. Seleccionamos una o varias regiones de detección en este espacio, y si un píxel cae en alguna de estas regiones, se considera una detección positiva (de piel). Sin embargo, en este caso, se pretende reducir el tamaño de los histogramas utilizados. Hasta ahora, hemos trabajado con histogramas de 256 niveles (8 bits). Para utilizar histogramas de la mitad de niveles (7 bits), es necesario realizar una cuantificación de los valores.
- ii. Para lograr esta cuantificación, se realizó una modificación en la función "algo1(points, dataset\_type)" para generar histogramas idénticos a los previamente creados, pero con la mitad de bins (128 niveles). Esta versión modificada de la función se denomina "algo1\_128(imagen, dataset\_type)." A continuación, se proporciona el código adicional y modificado. Se incluye únicamente el código pertinente, con contexto suficiente para entender su ubicación en relación al código general. Este código añadido se encarga de la cuantificación de los niveles Cb y Cr a 7 bits, así como de trazar líneas visuales en el histograma para delimitar las regiones diagonales que servirán como regiones de detección en este nuevo detector de histogramas de tamaño reducido.

```

1 % Recorremos cada imagen del directorio
2 for n = 1:length(image_files)
3     % Leemos la imagen
4     img = imread(fullfile(img_dir, image_files(n).name));
5
6     % Transformamos de RGB a YCbCr
7     img_ycbcr = rgb2ycbcr(img);
8
9     % Extraemos las componentes de crominancia Cb y Cr
10    img_cb = img_ycbcr(:,:,2);
11    img_cr = img_ycbcr(:,:,3);
12
13    quantized_matrix_cb = quantizePoints(img_cb, 128);
14    quantized_matrix_cr = quantizePoints(img_cr, 128);
15
16    % Leemos la máscara correspondiente
17    mask_name = strrep(image_files(n).name, 'jpg', 'bmp');
18    mask = imread(fullfile(mask_dir, mask_name));
19
20    % Aplicamos la máscara a las componentes Cb y Cr
21    img_mask_cb = quantized_matrix_cb(mask);
22    img_mask_cr = quantized_matrix_cr(mask);
23
24    % Concatenamos los vectores con datos de crominancia
25    vect_im_mask_cb = [vect_im_mask_cb; img_mask_cb];
26    vect_im_mask_cr = [vect_im_mask_cr; img_mask_cr];
27 end
28
29 % Creamos la matriz CbCr resultante
30 matriz_cbcr_final = [vect_im_mask_cb, vect_im_mask_cr];
31
32 % Creación y presentación del histograma en 3D
33 hist3(matriz_cbcr_final, 'Edges', {0:1:127, 0:1:127}, 'CDataMode', 'auto');
34 xlabel('Componente Cb');
35 ylabel('Componente Cr');
36 zlabel('Frecuencia');
37 title('Histograma de las Componentes de Crominancia Cb y Cr');
38
39 % Invertir los colores del colormap
40 colormap(flipud(colormap));
41 % Mostrar el gráfico
42 colorbar; % Agrega una barra de colores para mostrar la correspondencia de valores

```

```

1 % Dibujar líneas verticales y horizontales para marcar los límites
2 hold on;
3 %region 1
4 line([0, 127], [127, 0], 'Color', 'r', 'LineWidth', 2);
5 line([4, 127], [127, 4], 'Color', 'g', 'LineWidth', 2);
6 line([0, 114], [13, 127], 'Color', 'b', 'LineWidth', 2);
7 line([0, 127], [17, 81], 'Color', 'c', 'LineWidth', 2);
8
9 %region 2
10 line([7, 127], [127, 7], 'Color', 'r', 'LineWidth', 2);
11 line([127, 11], [11, 127], 'Color', 'g', 'LineWidth', 2);
12 line([0, 108], [19, 127], 'Color', 'b', 'LineWidth', 2);
13 line([0, 99], [28, 127], 'Color', 'c', 'LineWidth', 2);
14
15 %region 3
16 line([0, 127], [96, 32], 'Color', 'r', 'LineWidth', 2);
17 line([0, 125], [125, 0], 'Color', 'g', 'LineWidth', 2);
18 line([0, 113], [14, 127], 'Color', 'b', 'LineWidth', 2);
19 line([0, 103], [24, 127], 'Color', 'c', 'LineWidth', 2);
20 hold off;
21
22 % Agregar etiquetas a las líneas para mostrar los valores de los límites
23 text(76, 55, 'Region 1', 'Color', 'r', 'FontSize', 12);
24 text(58, 77, 'Region 2', 'Color', 'r', 'FontSize', 12);
25 text(55, 69, 'Region 3', 'Color', 'r', 'FontSize', 12);
26
27 end

```

- iii. Los únicos cambios implementados se encuentran en el bucle "for", que itera a través de cada imagen, extrayendo sus componentes Cb y Cr respectivos, aplicando la máscara ideal y guardando los píxeles

de piel resultantes en un vector específico para posteriormente construir el histograma. La misma transformación se ha aplicado tanto al canal Cb como al canal Cr. Para simplificar la explicación, en este punto, trabajaremos únicamente con el canal Cb.

- iv. En cuanto al código, hasta el momento en que se extraen los canales Cb y Cr de las imágenes, el código permanece completamente inalterado en comparación con la versión original. A partir de este punto, se introducen mejoras significativas. En primer lugar, se realiza la cuantificación de los canales a través de una nueva subfunción llamada "quantizePoints(image, histType)." El parámetro "image" debe consistir en una matriz 2D de dimensiones arbitrarias cuyos valores suelen ser uint8. En las siguientes demostraciones de código, se realizará una conversión de uint8 a double para permitir las operaciones internas necesarias. "histType" es un entero que puede tomar uno de los valores: 128, 64, 32 o 16. Si "histType" es igual a 256, lo que significa que no hay cuantificación, este código no debe utilizarse, ya que, si no se aplica cuantificación, el procedimiento es igual al original, que era sin cuantificación.
- v. El código para esta subfunción se presenta a continuación:

```

1 %Subfunction to quantize the points_matrix according to the wished upon hist. size
2 %
3 function quantized_matrix = quantizePoints(points_matrix, histType)
4 if nargin < 2
5     % Check if histType is not given (null)
6     quantized_matrix = points_matrix; % If not given, return the original matrix
7     return;
8 end
9
10 % Define the number of bits based on histType
11 switch histType
12     case 128
13         numBits = 7;
14     case 64
15         numBits = 6;
16     case 32
17         numBits = 5;
18     case 16
19         numBits = 4;
20     otherwise
21         error('Invalid histType value. Must be 128, 64, 32, or 16.');
22 end
23
24 % Define the quantization levels non-uniformly
25 numLevels = 2^numBits;
26 quantizationLevels = customQuantizationLevels(numLevels);
27
28 %Convert the following data to double precision: points_matrix
29 points_matrix = double(points_matrix);
30

```

```

31 % Quantize the points_matrix using non-uniform quantization
32 quantized_matrix = zeros(size(points_matrix));
33 for i = 1:size(points_matrix, 1)
34     for j = 1:size(points_matrix, 2)
35         % Find the closest quantization level
36
37         %conversion ratio from 256 levels to 128
38         proportion = points_matrix(i,j)/255;
39
40         [~, idx] = min(abs((127*proportion).*ones(1, numLevels) - quantizationLevels))
41         quantized_matrix(i, j) = quantizationLevels(idx);
42     end
43 end

```

- vi. Esta subfunción hace uso de otra subfunción llamada customQuantizationLevels(numLevels), cuyo código se da también aquí:

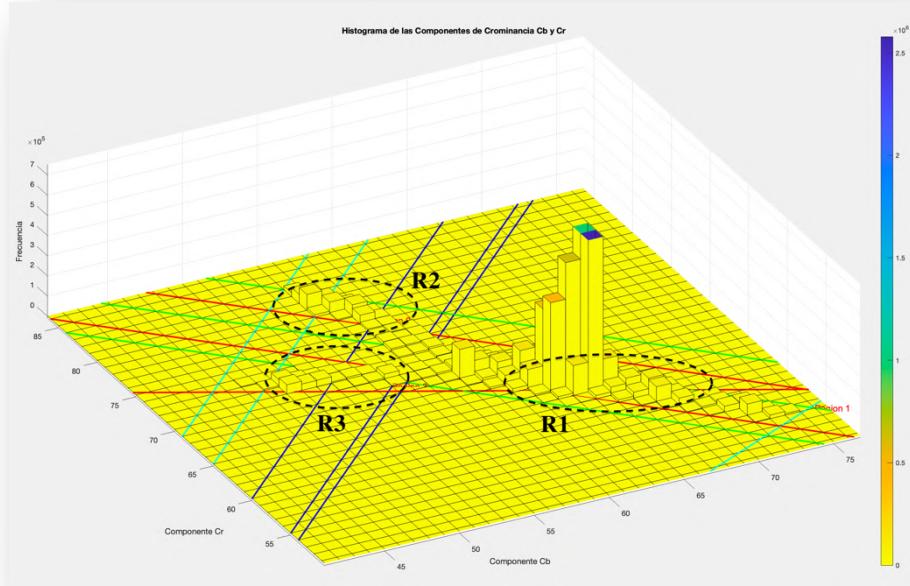
```

1 % Define custom non-uniform quantization levels
2 function levels = customQuantizationLevels(numLevels)
3 % Create more levels around the center and fewer levels towards the edges
4 center = 63; % Center of the [0, 255] interval
5 spread = 64; % Spread of levels around the center
6
7 % Distribute levels with more density around the center
8 levels = round(center + spread * linspace(-0.5, 0.5, numLevels));
9 levels(levels < 0) = 0; % Ensure levels are within [0, 255] range
10 levels(levels > 255) = 255;
11 end

```

- vii. Todas estas subfunciones tienen como propósito convertir los canales iniciales de Cb y Cr, que originalmente están en formato de 8 bits, a un nivel de bits inferior mediante el proceso de cuantificación. La salida de la primera subfunción, "quantizePoints," debe ser una matriz con las mismas dimensiones que la matriz de entrada o imagen de entrada. Sin embargo, en esta matriz de salida, los valores han sido cuantificados. Por ejemplo, si en la imagen de entrada se encuentra un píxel con un valor de Cb = 255 (el valor máximo), en la matriz de salida generada por esta función de cuantificación, ese mismo valor debería ser de Cb = 127 (la mitad), siempre y cuando se desee una cuantización de 7 bits (128 niveles).
- viii. Siguiendo el flujo del programa "algo1\_128," el funcionamiento sigue siendo el mismo. Se aplica la máscara lógica a las imágenes procesadas, que en este caso han sido cuantificadas, y se guardan sus valores en formato de vector columna para posteriormente dibujar su histograma, tal como se muestra en la interfaz del usuario.
- ix. Luego, se procede a trazar las líneas delimitadoras de las regiones de detección en el histograma. Los

valores de estas líneas se determinaron mediante cálculos simples y determinando la ecuación de la recta que pasaba por un par de puntos de manera que esta recta definiera una de las cuatro fronteras de la región en cuestión. Al finalizar la ejecución de este código en la función "algo1\_128," se genera el siguiente histograma:



- x. Como se aprecia directamente en la imagen, se muestran las tres regiones que utilizaremos para la detección de píxeles. A diferencia del detector original, que utilizaba cuatro regiones, donde podían caber píxeles no correspondientes a piel, la cuantificación ha dado lugar a tres regiones más diferenciadas de píxeles de piel. En este caso, trabajaremos con estas tres regiones en lugar de las cuatro utilizadas anteriormente.

#### - Puesta a Prueba del Detector Cuantificado

- i. A partir de este punto, teníamos una comprensión clara de la forma y el comportamiento de nuestro detector cuantificado. De manera similar al detector original, procesamos una imagen cuatro veces, cada vez representando el procesamiento de

una imagen en una región específica. Sin embargo, en este caso, solo trabajamos con tres regiones. La función encargada de realizar el procesamiento y el cálculo de las máscaras es la conocida función "algo2", que en este caso se convierte en "algo2\_128". Esta función toma dos entradas, "imagen" y "regions," al igual que en la versión original. Por lo tanto, el nuevo formato reconocido de la función es "algo2\_128(imagen, regions)." La variable "regions" debe ser una celda en Matlab que contiene toda la información necesaria para calcular la región de detección. A continuación, se muestra el código de esta función "algo2\_128":

```

1 function mask = algo2_128(imagen, region)
2
3 % Verificar si se proporcionaron los valores de Cb_min, Cb_max, Cr_min y Cr_max
4 if nargin < 2
5   error('Debes proporcionar un vector de puntos de control [Cb_min, Cb_max, Cr_min,
6 end
7
8 % Transformar la imagen a YCbCr
9 im_ycbcr = rgb2ycbcr(imagen);
10
11 %extract Cb and Cr components
12 Cb = im_ycbcr(:,:,2);
13 Cr = im_ycbcr(:,:,3);
14
15 %quantize final image to be thresholded
16 Cb_quantized = quantizePoints(Cb, 128);
17 Cr_quantized = quantizePoints(Cr, 128);
18
19 %concatenate these 2 matrices together
20 img_filtered = cat(3, Cb_quantized, Cr_quantized);
21
22 %compute mask
23 mask = checkBetweenLines(img_filtered, region);
24 end

```

- ii. Esta función utiliza dos subfunciones: la mencionada anteriormente "quantizePoints(imagen, histType)" y la aún no comentada "checkBetweenLines(imagen, region)". La variable "imagen" representa una matriz 2D con los valores Cb o Cr de los píxeles (del mismo tamaño que la imagen original), mientras que "region" contiene el conjunto de información mencionado en el párrafo anterior.
- iii. El comportamiento de esta función es idéntico al del "algo2" original, pero en este caso, hemos tenido que utilizar una función de detección interna, "checkBetweenLines()", que es mucho más compleja. Esto se debe a que las regiones de detección dejan de tener una forma geométrica tan sencilla como en el detector original. Algo que

hemos querido implementar como nuevo sistema para así aprovechar al máximo el potencial de esta posible mejora. Además, con la cuantificación, las regiones de los píxeles de piel adoptan una forma incluso más diagonal y estrecha que la original, como se puede observar en el histograma de la página anterior. Por lo tanto, trabajar con regiones romboidales o diagonales, o simplemente rectángulos y cuadrados girados con respecto a los ejes, podría facilitar la detección.

- iv. En resumen, la variable de entrada "region" debe contener las pendientes (m) y los y-intercept (n) de las cuatro rectas que delimitan la región de detección. Es dentro de la función "checkBetweenLines" donde ocurre la verificación. A continuación, se muestra el código de esta función:

```

1 %function to test whether a point is situated between 2 lines (different detection al
2 function isBetween = checkBetweenLines(matrix, region)
3 [rows, cols] = size(matrix);
4 isBetween = zeros(rows, cols/2);
5
6 %extract line regions from region input
7 Extract1 = region{1};
8 Extract2 = region{2};
9 Extract3 = region{3};
10 Extract4 = region{4};
11
12 %slope & y-intercept assignations
13 m1 = Extract1(1);
14 n1 = Extract1(2);
15
16 m2 = Extract2(1);
17 n2 = Extract2(2);
18
19 m3 = Extract3(1);
20 n3 = Extract3(2);
21
22 m4 = Extract4(1);
23 n4 = Extract4(2);
24
25 for i = 1:rows
26     for j = 1:cols/2
27         x = matrix(i, j, 1);
28         y = matrix(i, j, 2);
29
30         line1 = m1*x + n1;
31         line2 = m2*x + n2;
32         %line3 = m3*x + n3;
33         line3 = cell((y-n3)/m3);
34         %line4 = m4*x + n4;
35         line4 = cell((y-n4)/m4);
36
37         if (y <= line1 && y >= line2 && x >= line3 && x <= line4)
38             isBetween(i, j) = 1;
39         else
40             isBetween(i, j) = 0;
41         end
42     end
43 end
44 end

```

- v. Al inspeccionar detenidamente esta función, se observará que su propósito, como se mencionó anteriormente, es detectar si un valor de Cb o Cr se encuentra dentro de una región geométrica delimitada por cuatro rectas en dicho espacio de color. Si el píxel está dentro de la región, se le asigna el valor 1, y si no, se le asigna el valor 0. Al final de la ejecución, obtenemos una máscara de ceros y unos para esa región. Este procedimiento debe repetirse un total de tres veces, ya que debemos realizar la verificación en tres regiones distintas. El algoritmo original se llamaba "algo3(points\_matrix, dataset\_type, ...)", y ahora se llama "algo3\_128", con las mismas entradas, excepto el argumento de "points\_matrix" del "algo3" original. Esto se debe a que el cálculo de la región de detección es complicado (a través de los coeficientes m y n de las rectas delimitadoras), y se decidió declarar esta región dentro del algoritmo, abstrayendo la complejidad de este cálculo del usuario. No obstante, esto puede cambiarse fácilmente, permitiendo al usuario insertar su región de detección preferida en la función "algo3\_128".
- vi. En resumen, este algoritmo es idéntico al original, pero dentro de él se declara la región de detección (que en realidad contiene tres regiones diferentes, ya que realizamos la verificación en tres ubicaciones diferentes). También se actualizó la llamada a la función "algo2" en la subfunción "generateMasks(image, points\_matrix)", para que llame a la nueva función "algo2\_128". Sin embargo, estos son todos los cambios, ya que los nuevos cálculos se realizan dentro de la función "algo2\_128". "Algo3" es simplemente responsable de aplicar otras técnicas de procesamiento que ya se han detallado, como la eliminación de componentes pequeñas, los tratamientos con aperturas, etc.
- vii. A continuación, se muestran solo las líneas de código añadidas en la extensión de esta nueva función "algo3\_128":

```

1 % Vector of cell arrays containing line parameters for each region
2 % (4 lines per region)
3
4 first_region = {[[-1, 131], [-1, 127], [0.5, 17.5], [1, 13]}];
5 second_region = {[[-1, 138], [-1, 134], [1, 28], [1, 19]}];
6 third_region = {[[-1, 125], [-0.5, 96], [1, 24], [1, 14]}];
7 points_matrix = {first_region, second_region, third_region};

```

```

1 function masks = generateMasks(image, points_matrix)
2 % Utilizar algo2 para generar máscaras para las regiones
3 masks = arrayfun(@(i) algo2_128(image, points_matrix{i}), ...
4     1:size(points_matrix, 1), 'UniformOutput', false);
5 end

```

- viii. Como se muestra en las figuras anteriores, en la función "algo3\_128" se declaran las regiones de detección, que luego se pasan a la función "generateMasks()" (ubicada justo debajo de la primera figura). Esta función llama a la función "algo2\_128" tres veces, procesando la imagen de entrada con cada una de las tres regiones mediante la función integrada en Matlab "arrayfun()". Una vez obtenemos las tres máscaras distintas, se superponen realizando una operación OR lógica entre las tres. El funcionamiento es esencialmente idéntico al del "algo3" original, por lo que recomiendo consultar los apartados pertinentes de este documento si desea obtener más información sobre el funcionamiento del "algo3" original.

## - Resultados Obtenidos

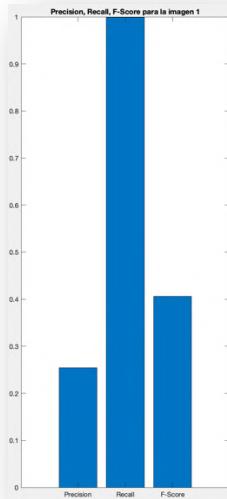
- i. Como se expondrá a continuación, los resultados obtenidos mediante esta arquitectura de histograma reducido y cuantificado, hasta el momento y según los experimentos realizados, no han arrojado conclusiones satisfactorias. La ejecución del "algo3\_128", seguida de la ejecución del "algo4\_128" (que, en realidad, es exactamente el mismo algoritmo que el "algo4" original y podría llamarse de la misma manera), produjo valores de F-Score muy inferiores, alrededor de 0.5, en comparación con los F-Scores obtenidos mediante el uso de histogramas de tamaño completo (8 bits/256 niveles). A continuación, se muestra un ejemplo de una imagen tomada del "Training Dataset" junto con la máscara generada mediante los nuevos algoritmos "algo2\_128" y "algo3\_128"

(donde los píxeles detectados como positivos se marcan como 1, es decir, en blanco). Además, se incluye un gráfico que muestra los valores de Recall, Precision y el F-Score final para esa imagen procesada. Si bien se muestra solo una imagen, la tendencia se repite para otras imágenes, es decir, los resultados eran prácticamente idénticos para otras imágenes.



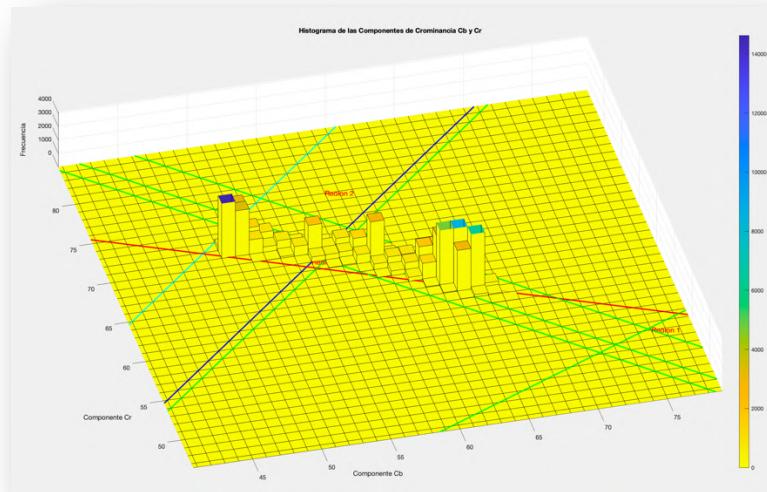
Imagen original 1 escogida

Máscara computada

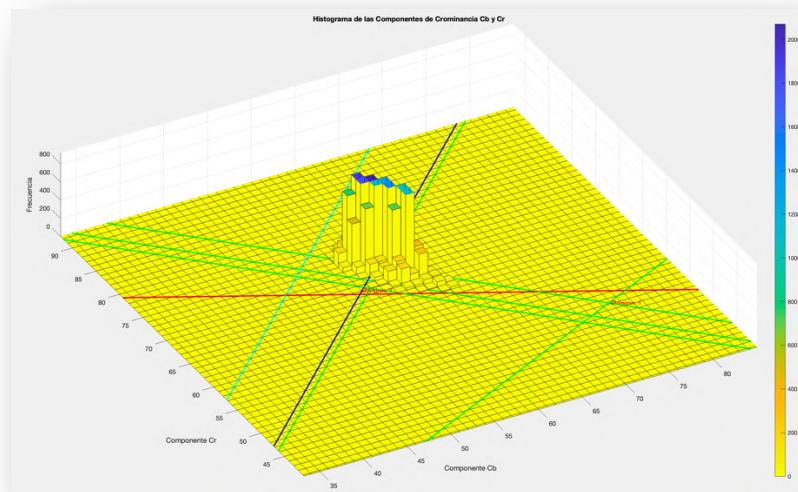


Valores de Precision, Recall y F\_Score

- ii. Una explicación sencilla de estos resultados insatisfactorios se puede fundamentar en la siguiente representación de histogramas que muestra la distribución de píxeles de piel y no piel en el espacio cromático Cb y Cr después de la cuantificación. El efecto es evidente y demuestra ser el principal problema al reducir la cuantificación de los histogramas a niveles inferiores.



**Histograma de Píxeles de Piel de Imagen 1**



**Histograma de Píxeles de No Piel de Imagen 1**

- iii. Como se puede apreciar, las regiones seleccionadas para la detección de piel, aunque parecen ser adecuadas para la imagen 1 al detectar una gran cantidad de píxeles de piel, también resultan en la detección de píxeles de no piel en grandes cantidades. Esto se evidencia claramente en el último de los dos histogramas, donde muchos píxeles de no piel se encuentran en la región superior izquierda y la región general de la derecha del histograma. En la gráfica que muestra los valores de Recall, Precision y F\_Score, esto se traduce en un alto valor de Recall, ya que se detectan muchos píxeles de piel. Sin embargo, este alto Recall no refleja un buen rendimiento del detector, ya que también se detectan numerosos píxeles de no piel. Esto da como resultado un valor de Precision considerablemente bajo, lo que afecta negativamente al F\_Score, resultando en un resultado deficiente.
- iv. En resumen, este comportamiento se debe a la cuantificación, que causa que tanto píxeles de piel como de no piel caigan en los mismos "bins" o posiciones en el histograma. Si bien en un principio la idea parecía beneficiosa, pues nos permitía generar regiones más pequeñas y por tanto más fáciles de delimitar, la falta de distinción entre estos tipos de píxeles lleva a un mayor número de detecciones incorrectas, lo que se refleja en los bajos valores de F\_Score obtenidos.
- v. Debido a estos resultados insatisfactorios, decidimos no continuar experimentando con histogramas de 64 niveles, 32 niveles o 16 niveles, ya que la cuantificación de 128 niveles ya proporcionó resultados lo suficientemente bajos.
- vi. A pesar de que este intento no haya resultado en mejoras concretas en el sistema, ha proporcionado claridad y valiosas lecciones en el proceso de investigación. Este enfoque ha demostrado que la precisión es esencial para lograr resultados óptimos en la detección de píxeles. La generalización hacia regiones más simples o la simplificación excesiva de la información disponible resultan contraproducentes. La solución más efectiva es

mantener un enfoque detallado y definir los píxeles de la manera más precisa posible.

- vii. A través de herramientas matemáticas y enfoques empíricos que cuantifican de manera precisa las áreas de detección óptimas, así como el posterior procesamiento de las máscaras generadas, hemos aprendido que los algoritmos diseñados para simplificar en exceso terminan generando máscaras de poco valor práctico. La clave está en mantener el nivel de detalle necesario para una detección precisa y eficaz.
- viii. El proceso de detección de piel es una tarea meticulosa que requiere una atención especial si se pretenden alcanzar resultados precisos y algoritmos altamente generalizables. La amplia investigación y desarrollo presentados en este informe, asimismo como nuestros excelentes resultados, respaldan esta afirmación de manera contundente.

## **Nota del autor**

---

Aquí concluye el informe del Sistema I para detección de piel, no obstante, antes de finalizar, me gustaría compartir algunos comentarios en calidad de autor de este proyecto.

A lo largo de este proceso, tanto en las etapas iniciales de investigación como en el proceso de mejora del sistema, se ha llevado a cabo una extensa labor de indagación y evaluación de opciones. Cabe mencionar que las únicas fuentes de referencia utilizadas en este trabajo han sido documentos académicos y de investigación que aportaran bases objetivas previas al desarrollo. Sin embargo, todas las afirmaciones y decisiones contenidas en este informe se basan únicamente en los resultados obtenidos mediante nuestra propia investigación, los cuales se han detallado minuciosamente a lo largo de este informe. A excepción de unas pocas afirmaciones comentadas en la introducción y añadidas en las referencias.

El desarrollo de este sistema implicó la exploración de diversas alternativas, desde sistemas y algoritmos prometedores hasta aquellos que, en última instancia, se revelaron ineficientes o poco exitosos, siendo descartados en el proceso. El presente informe es el fruto de los métodos e ideas que demostraron ser efectivas y beneficiosas, reflejando los frutos de esta selección.

Si bien este informe puede parecer extenso, hemos hecho un esfuerzo deliberado por explicar en detalle cada paso del proceso de pensamiento y las opciones planteadas. Nuestra intención ha sido permitir al lector una comprensión completa de la trayectoria que condujo a cada uno de los resultados y, por tanto, al consiguiente sistema resultante. Quiero agradecer al lector por dedicar su tiempo a evaluar nuestra propuesta, y espero que esta lectura haya sido tan enriquecedora para usted como lo fue para nosotros este proceso de investigación, aprendizaje y desarrollo.

Agradezco su atención y consideración en nuestro trabajo.

**Pol Buitrago Esteve**

## **Referencias**

---

<sup>1</sup> UPC. "Procesado de Imagen y Video, Otoño 2022. Diseño, Implementación y Evaluación de algoritmos de procesado de imagen." Asignatura PIV, Propuesta Prog1, 2022.

<sup>2</sup> Ahmed Elgammal, Crystal Muang, y Dunxu Hu. "Skin Detection - a Short Tutorial." Departamento de Ciencias de la Computación, Universidad de Rutgers, Piscataway, NJ, 08902, USA.