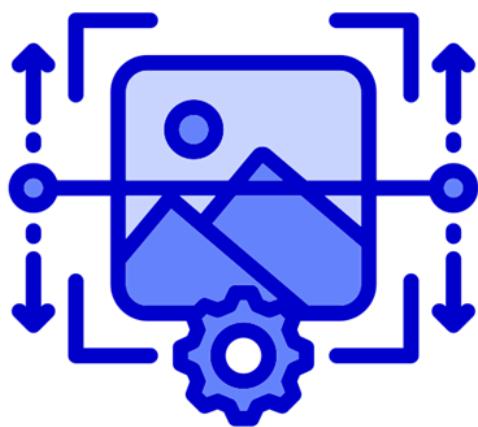




## TP2 - Procesamiento de Imágenes 1

### Tecnicatura Universitaria en Inteligencia Artificial.



#### Docentes:

- Gonzalo Sad
- Julián Alvarez
- Juan Manuel Calle

#### Integrantes (Grupo 11):

- Marcela Flaibani  
F-3793/1
- Facundo López Crespo  
L-3339/1
- Daniela Dito  
D-4322/2

## Índice

<b>Enunciados.....</b>	<b>3</b>
<b>Resolución.....</b>	<b>5</b>
Ejercicio 1 .....	5
Ejercicio 2 .....	12
<b>Repository.....</b>	<b>25</b>

## Enunciados

### **Problema 1 - Detección y clasificación de monedas y dados**

En la Figura 1 se muestra la imagen del archivo *monedas.jpg*, la cual consiste en un **fondo de intensidad no uniforme** sobre el cual se hallan **dados** y **monedas** de distinto valor y tamaño.

Se debe elaborar un algoritmo capaz de procesar dicha imagen y resolver los siguientes puntos sobre ella:

- A. Procesar la imagen de manera de segmentar las monedas y los dados de manera **automática**.
- B. **Clasificar** los distintos tipos de monedas y realizar un conteo automático.
- C. **Determinar** el número del valor que representa la cara superior de cada dado y realizar un conteo automático.

**AVISO:** En cada punto, el *script* elaborado debe informar y mostrar los resultados en cada una de las etapas de procesamiento.



## **Problema 2 - Detección de patentes**

En la Figura 2 se muestra una de las doce imágenes (archivos img<id>.png), las cuales representan la vista anterior o posterior de diversos vehículos. En cada una de ellas se puede visualizar las correspondientes patentes.

Se debe elaborar un algoritmo capaz de procesar cada una de estas imágenes y resolver los siguientes puntos en cada uno de ellas:

- A. **Detectar automáticamente** la placa patente y segmentar la misma. Informar las distintas etapas de procesamiento y mostrar los resultados de cada etapa.
- B. Implementar un algoritmo de procesamiento que **segmente** los caracteres de la placa patente detectada en el punto anterior. Informar las distintas etapas de procesamiento y mostrar los resultados de cada etapa.



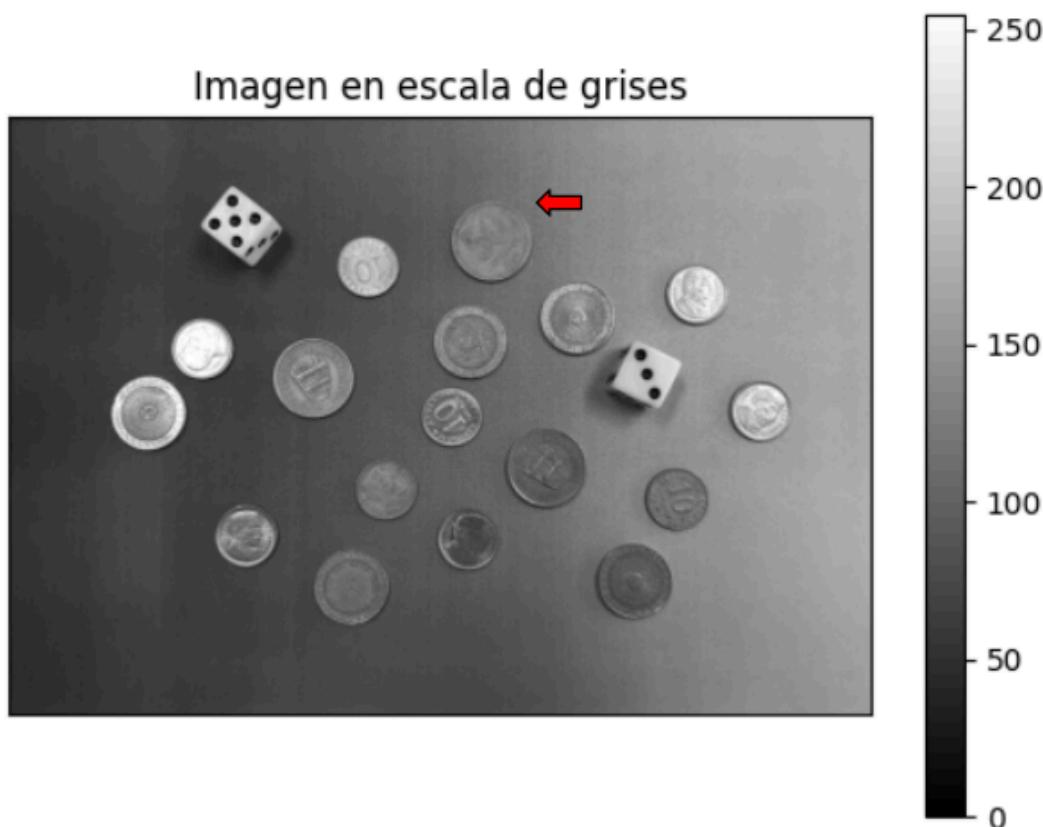
Figura 2: Ejemplo de una de las imágenes (archivo *img05.png*) de la carpeta *Patentes*.

## Resolución

### Ejercicio 1: Monedas y dados

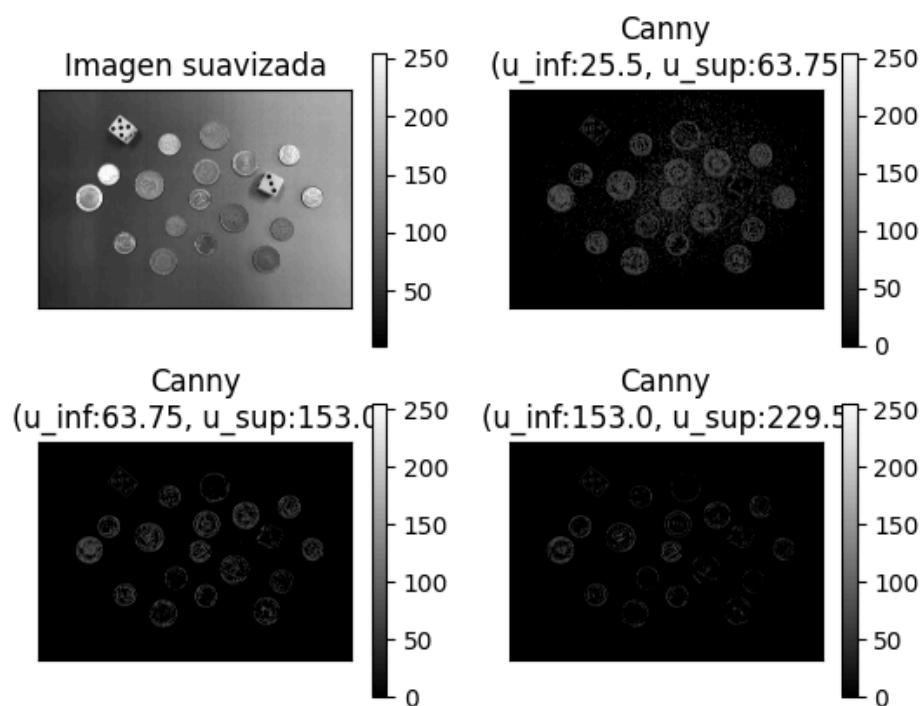
#### Enunciado A

La mayor dificultad que presenta la segmentación de estos objetos es el bajo contraste entre los valores de los píxeles pertenecientes a los objetos y los valores de los píxeles pertenecientes al fondo de la imagen, llegando incluso a identificarse zonas en las cuales los cambios de intensidad son significativamente bajos. Con una flecha roja se indica un ejemplo de zona en la imagen que presenta esta característica.



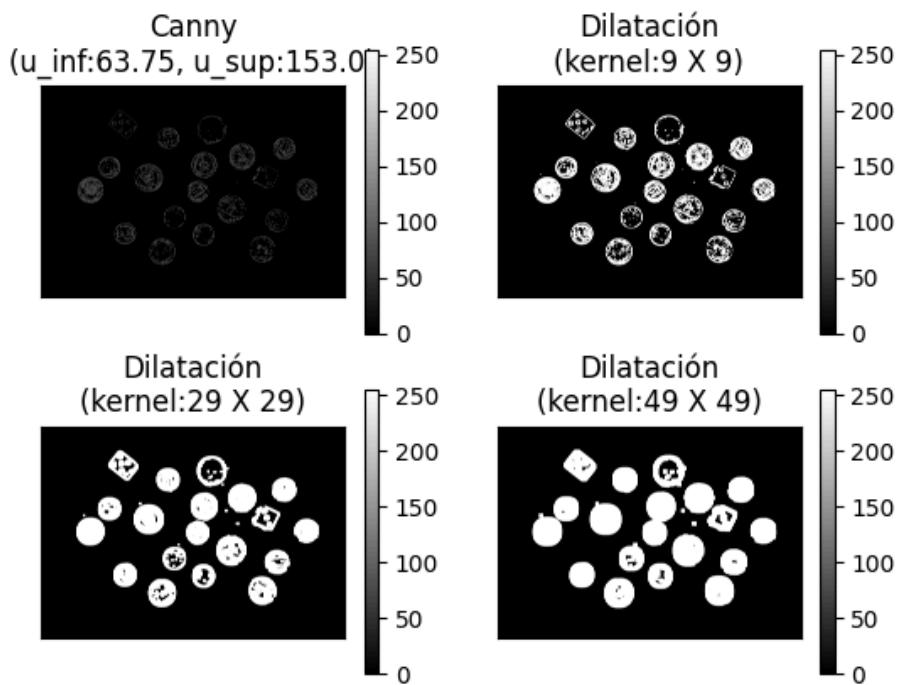
Esta particularidad implica que no pueda utilizarse un valor de pixel que sirva de umbral para separar los píxeles que pertenecen a los objetos de aquellos que pertenecen al fondo de la imagen. Por tal motivo, se aplican técnicas de procesamiento de imágenes alternativas.

La primera técnica que se utiliza es Canny con el objetivo de alcanzar una imagen binaria en la cual los bordes de los objetos presenten como valor de píxel 255. Se prueban distintos valores de umbrales, recordándose que esta técnica utiliza doble umbralado, alto y bajo, por lo cual, se utilizan parejas de umbrales.

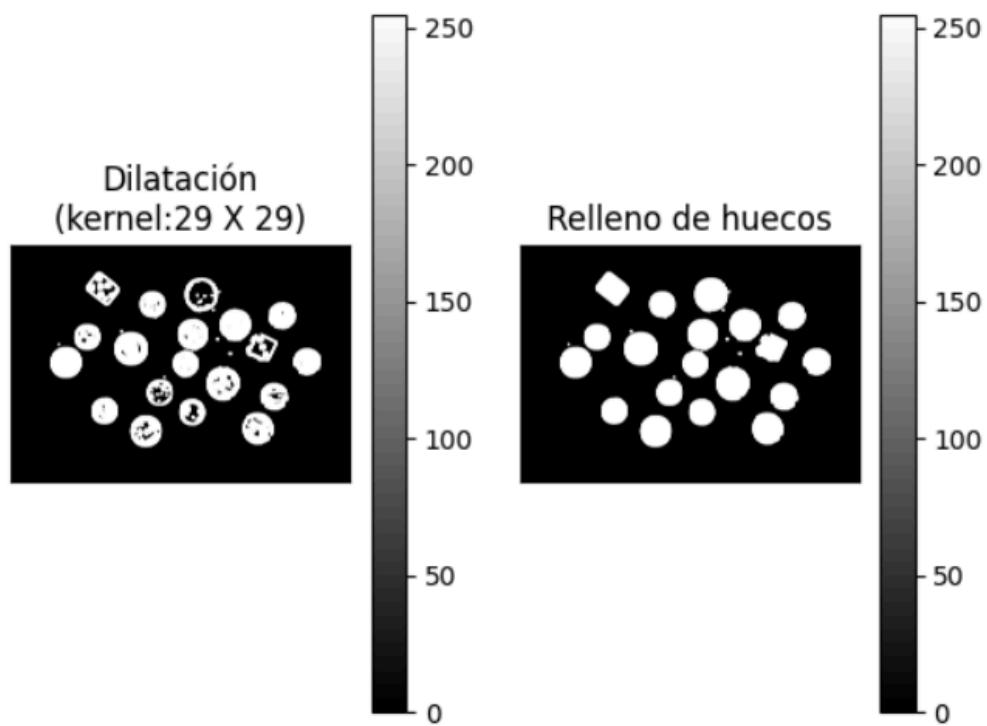


Se opta por la alternativa intermedia porque es aquella que lleva a cero la mayoría de los píxeles que pertenecen al fondo de la imagen y a su vez detecta una buena cantidad de píxeles que pertenecen a los objetos de la imagen. No obstante ello, la técnica referida es aún insuficiente para lograr la segmentación completa. Por ello se emplea una segunda técnica de procesamiento de imágenes vinculada con el procesamiento morfológico.

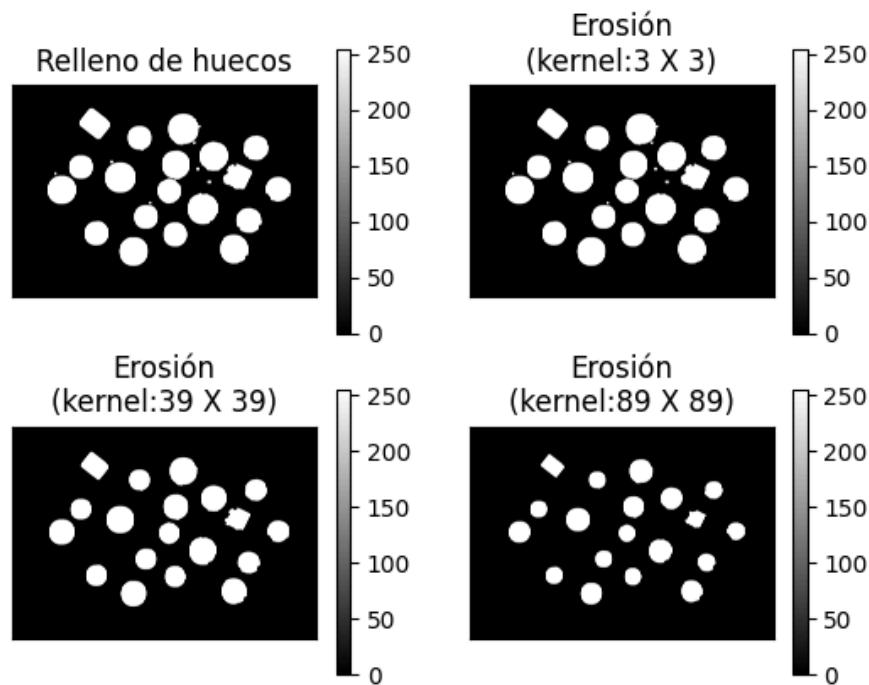
La primera operación que se aplica es la de la Dilatación con el objetivo de engrosar los bordes de los objetos, en vista de lograr que pueda completarse el cortorno de cada objeto, para lo cual se emplean distintos tamaños de máscara.



La alternativa intermedia es la que ofrece el mejor resultado porque permite cerrar los bordes de los objetos sin que éstos lleguen a aparearse entre sí. Una vez que se alcanza esta condición cobra sentido rellenar los objetos utilizando como operación la Reconstrucción Morfológica.



Todavía en esta etapa del procesamiento de la imagen hay ruido, los contornos de los objetos distan de ser suaves como así también existe cierto grado de solapamiento entre ellos<sup>1</sup>. Por estas razones, se justifica utilizar la operación de Erosión.



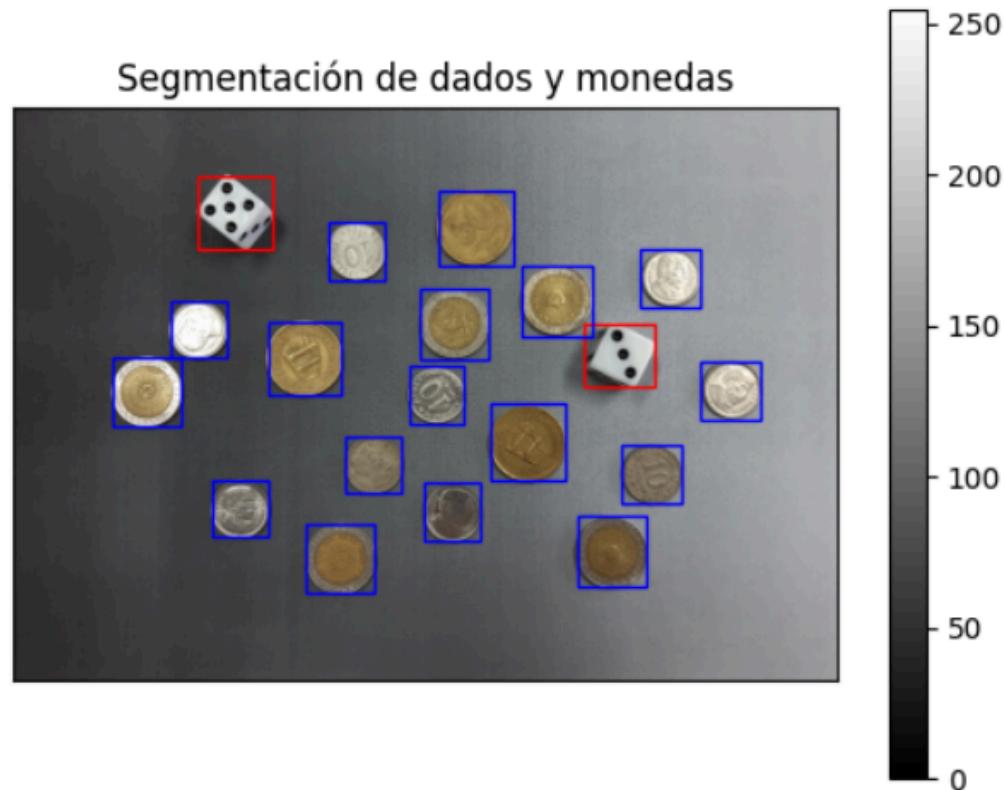
Entre los distintos tamaños probados para el elemento estructural utilizado en esta operación, el de la opción intermedia es el que mejor resultados reporta porque permite eliminar el ruido en la imagen evitando que los objetos se distorsionen significativamente.

Una vez alcanzado este estadio, se aplica el algoritmo Componentes Conectadas, el cual permite identificar el bounding box de cada objeto, para luego utilizar Encontrar Contornos. Una vez identificado el contorno de un objetos, puede estimarse tanto su área como su perímetro con funciones propias de OpenCV. Luego, mediante el análisis de su geometría, se aproxima su factor de forma.

Al conocerse que el factor de forma propio de un círculo es 0.0796, se infiere que, si el factor de forma del objeto bajo análisis es inferior a 0.0657, por alejarse significativamente de la forma de un círculo, el objeto en cuestión es un dado, y en caso contrario, es una moneda. Por lo cual, este umbral permite segmentar diferenciar perfectamente los objetos entre dados y monedas.

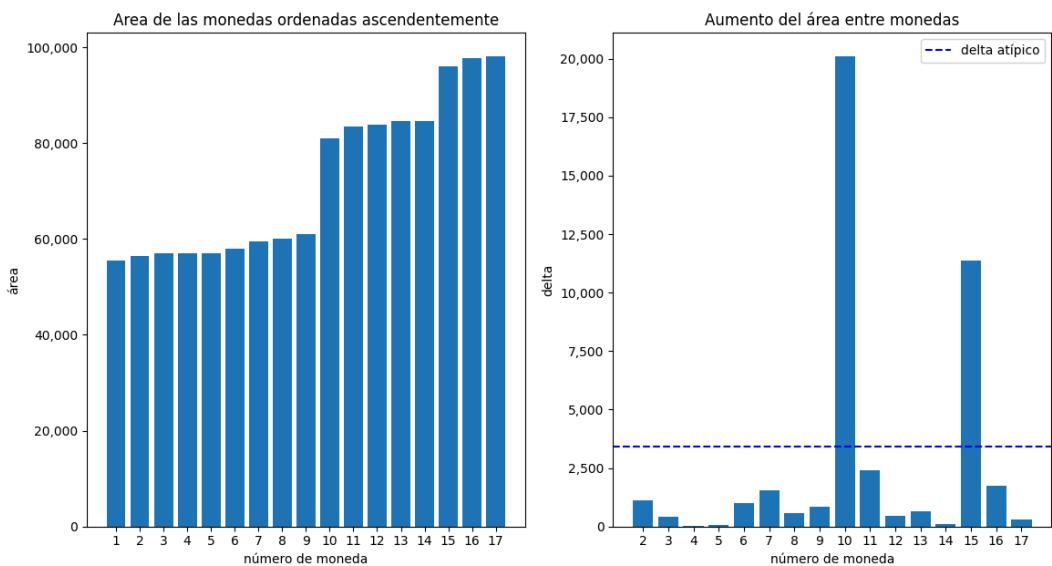
---

<sup>1</sup> Se verifica que cuando en el bounding box obtenido con Componentes Conectadas aparecen segmentos de otros objetos, el área calculada no es la correspondiente al objeto principal, sino que se corresponde con el segmento del otro objeto que se solapa.

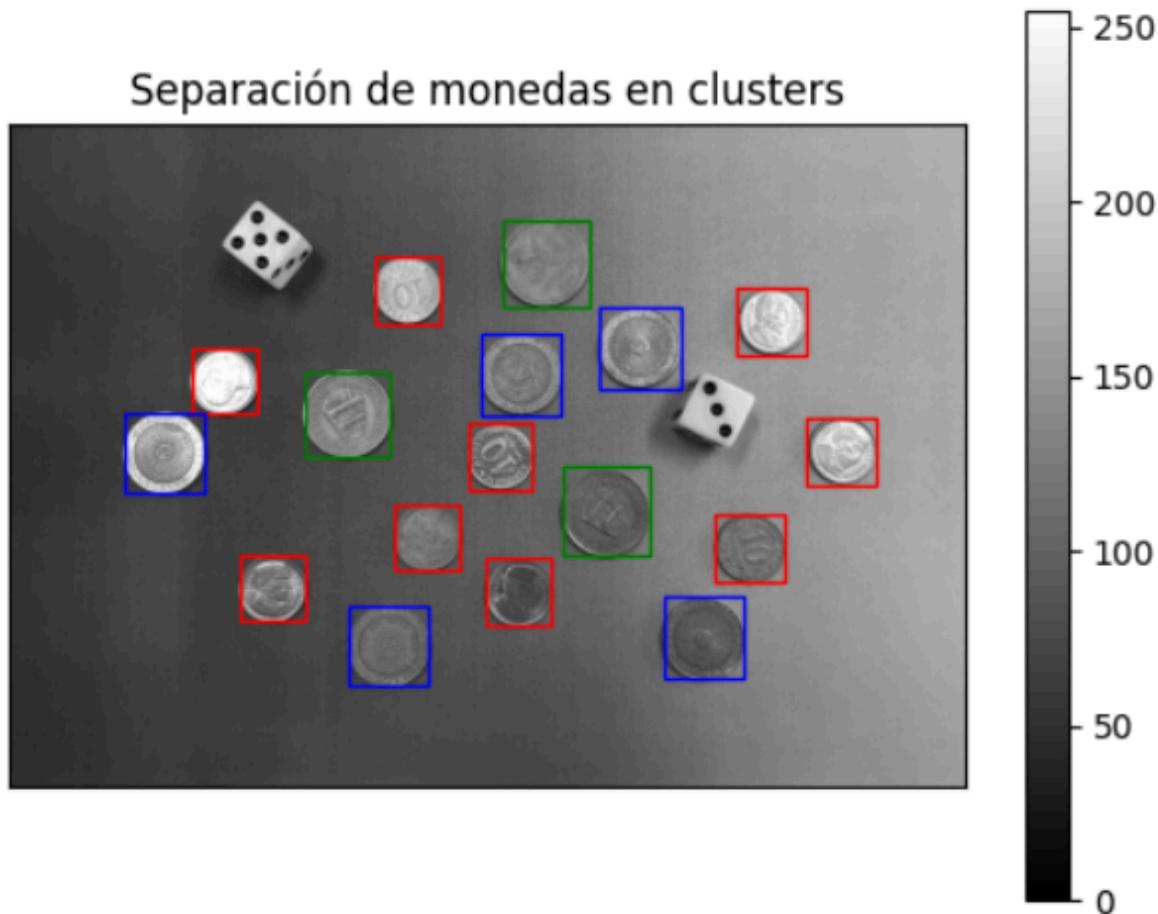


### Enunciado B

Para la clasificación de las monedas se analizan sus respectivas áreas bajo la premisa que aquellas monedas que son del mismo tipo deben presentar áreas similares. Por tal motivo, en una primera instancia, se ordenaron las monedas según su superficie, de menor a mayor, y en una segunda instancia, se calcula el aumento del área al pasar de una moneda a la otra. En aquellos casos en que se distingue que este delta es particularmente alto, se infiere que se trata de un nuevo tipo de moneda.



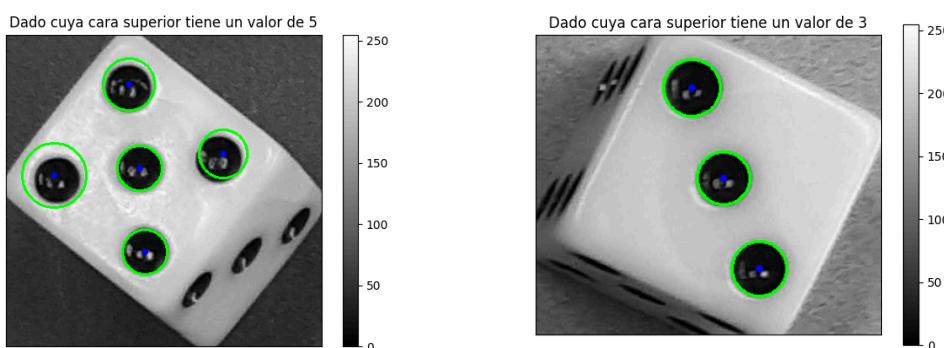
La moneda número diez presenta un delta que supera largamente el umbral<sup>2</sup>, por lo cual, puede inferirse que hay un segundo tipo de monedas, y con la moneda número 15 ocurre lo mismo, por lo cual, puede inferirse que hay un tercer tipo de monedas.



<sup>2</sup> Este umbral que se denomina delta atípico o delta separador de clusters es calculado a partir de esta fórmula:  $q3 + RI * 1.5$ . Donde q3 es el cuartil 3 de los deltas y RI es el rango intercuartil de esta variable.

### Enunciado C

Para distinguir el valor de la cara superior de los dados se utiliza la Transformada de Hough, más precisamente el algoritmo HoughCircles que provee la librería OpenCV para detectar círculos en imágenes. Este algoritmo se aplica sobre el cropping de cada dado para garantizar que la cantidad de círculos detectados se corresponda con los círculos que se circunscriben a la cara superior del dado. Se infiere que el conteo de los círculos encontrados por el algoritmo se corresponde con el valor del dado, y a su vez, al devolver este algoritmo, para cada círculo encontrado, la coordenada del centro y el radio, se procede a mostrar los círculos que se encontraron.



### Resultado final



## Resolución

### Ejercicio 2 : Detección de patentes

#### 1 - Pre-procesamiento de las imágenes

```
def preprocess_image(img: np.ndarray, umbral: int) -> np.ndarray:
```

1. La imagen original en escala de grises, se binariza con un umbral. Este umbral varía en cada ejecución hasta encontrar el apropiado para la detección de la patente.
2. A la imagen original en escala de grises, se aplica una transformación Top Hat para lograr más nitidez y luego se binariza. Se utiliza un elemento estructural elíptico de 7 x 7.
3. Luego se realiza la intersección entre ambas imágenes binarias. La intersección actúa como un filtro que integra lo mejor de ambos enfoques.

El proceso del punto 1 alcanza para identificar la mayoría de las patentes, con un determinado umbral.

La siguiente tabla muestra en verde las imágenes identificadas y en rojo la que no fue identificada.

La primer columna corresponde al proceso 1, y las demás al proceso completo, con distintos umbrales para binarizar la imagen generada después de aplicar la transformación Top Hat.

El umbral para binarizar la imagen Top Hat puede variar entre 45 y 57. Se muestra como disminuye la cantidad de iteraciones necesarias para identificar una imagen, al aumentar el valor de ese umbral.

Se intentó usar un umbral dinámico con cv2.THRESH\_OTSU, pero selecciona un umbral muy bajo porque la mayoría de los píxeles en la imagen tienen valores de intensidad bajos. Falla con las imágenes 1 y 3.

Imagen	Umbralado		Umbralado + Top Hat Umb.		Umbralado + Top Hat Umb.		Umbralado + Top Hat Umb.	
			Umbral TH = 50		Umbral TH = 55		Umbral TH = 57	
	Umbral	Iteraciones	Umbral	Iteraciones	Umbral	Iteraciones	Umbral	Iteraciones
1	146	36	110	1	110	1	110	1
2	110	1	110	1	110	1	110	1
3			110	1	110	1	110	1
4	122	12	116	6	116	6	110	1
5	110	1	110	1	110	1	110	1
6	120	10	114	4	110	1	110	1
7	110	1	110	1	110	1	110	1
8	126	16	124	14	124	14	124	14
9	110	1	110	1	110	11	110	1
10	112	2	112	2	112	2	112	2
11	134	24	134	24	134	24	134	24
12	110	1	110	1	110	1	110	1
		105		57		64		49

Debido a la reducción de iteraciones, se elige aplicar el mismo preprocesamiento a todas las imágenes, en lugar de hacer un tratamiento morfológico particular para la imagen 3.

Cabe aclarar que el programa busca identificar una secuencia de caracteres que se ajusten al formato de una patente y no segmenta el borde de la misma.

Se trabaja con el área de la imagen original y no con el sector propio de la patente. Por lo tanto, el número de iteraciones con distintos umbrales puede deberse, tanto a que los componentes estén 8-conectadas con algún borde o entre sí, como a que ya estén individualizados pero alguna de las distancias entre ellos esté fuera de los rangos de separación calculados.

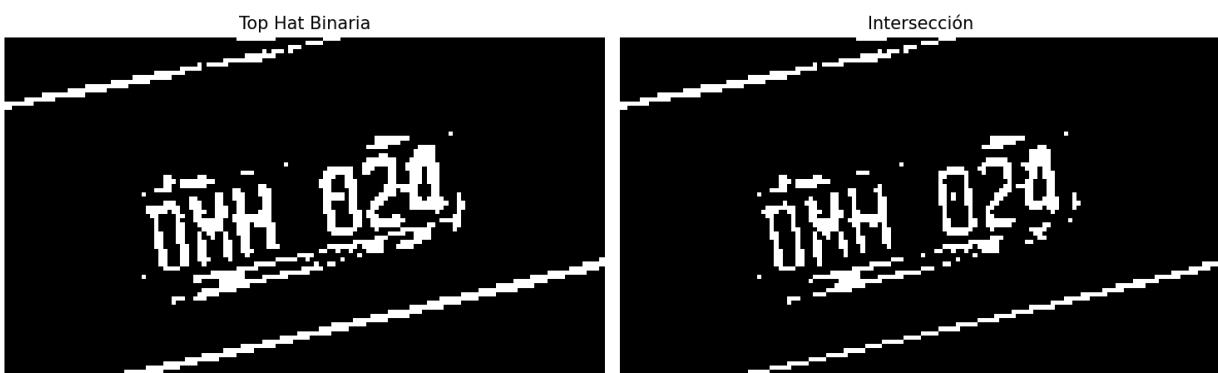
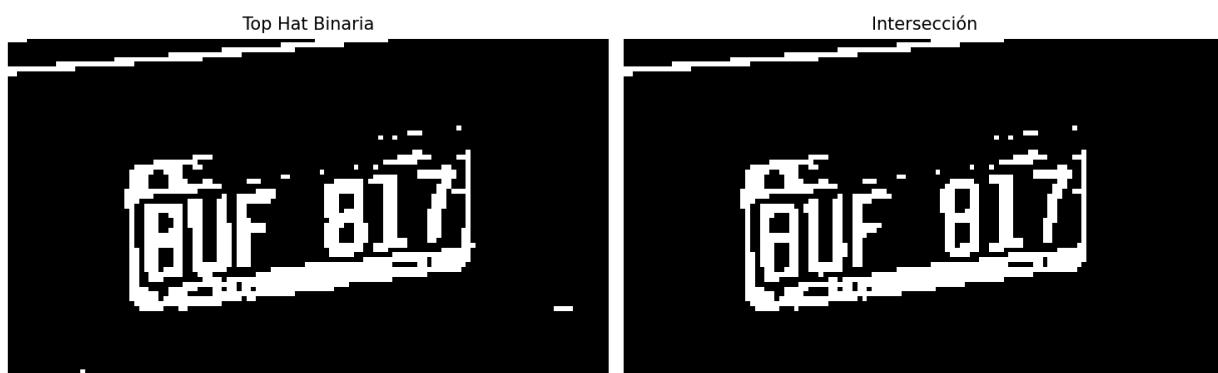
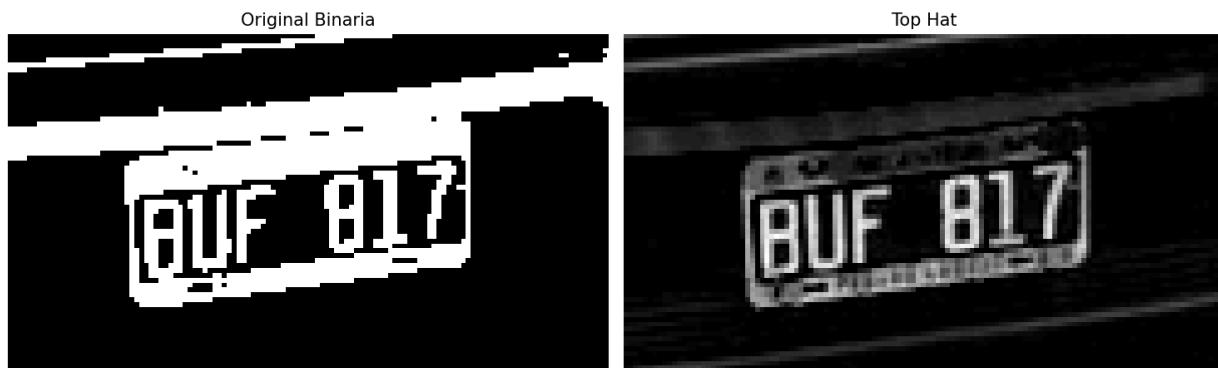
Las imágenes muestran el proceso:

```
def show_preprocessing_results(bynary_img, top_hat, binary_top_hat_img,
intersection)-> None:
```

Imagen que puede segmentarse sólo con el umbralado de la imagen a escala de grises. El preprocesamiento no afecta su segmentación.



Imágenes que pueden segmentarse luego del preprocesamiento.



Ej.: en la imagen 01 el carácter toca el borde inferior izquierdo después de la binarización.  
Si se aumenta el umbral se discontiñúa la letra D.

La imagen resultante de la transformación Top Hat umbralizada (imagen 03) no toca ese borde, pero los caracteres se engrosan y dos se tocan entre sí. La intersección (imagen 04) de las dos imágenes soluciona el problema.

## 2 - Filtro por área y relación de aspecto

```
def filter_area_aspect(img: np.ndarray) -> np.ndarray:
```

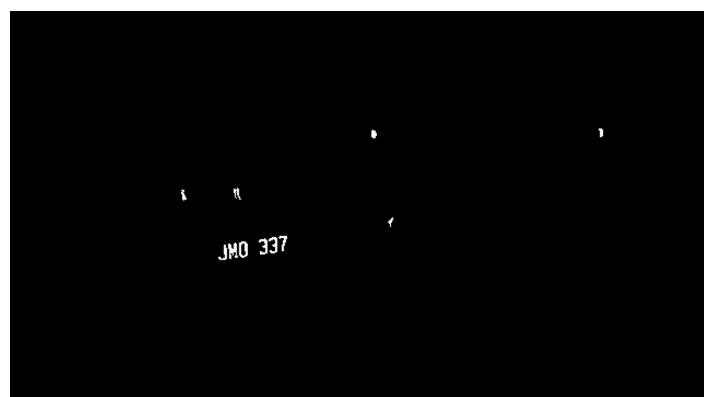
En la imagen resultante del preprocessamiento, se filtran los componentes conectados que no cumplen con los siguientes criterios:

- Área entre min\_area y max\_area.
- Relación de aspecto (altura/ancho) entre 1.5 y 3 (proporciones de los caracteres de una patente).

Sólo los componentes que cumplen estos criterios, se mantienen en la imagen de salida.

Se utiliza cv2.connectedComponentsWithStats para encontrar los componentes conectados en la imagen. Este método devuelve las estadísticas de cada componente, como su área, altura, y ancho.

Los resultados se muestran en las imágenes.



El umbral no alcanza para aislar los caracteres.



Imagen filtrada resultante



Se alcanza el umbral adecuado.

Imagen filtrada resultante



### 3 - Detección de patente

`def detect_patente(img: np.ndarray) -> tuple:`

Las patentes tienen un formato regulado.

El formato de estas patentes estuvo vigente desde 1995 hasta 2015.

❖ Formato

"ABC 123", es decir:

Tres letras (por ejemplo, "ABC").

Un espacio.

Tres números (por ejemplo, "123").

❖ Normas sobre dimensiones y espaciado

\*Tamaño de la placa:

Dimensiones de la placa: 400 mm x 130 mm (aproximadamente).

\*Tamaño de los caracteres:

Altura de las letras y números: 60 mm.

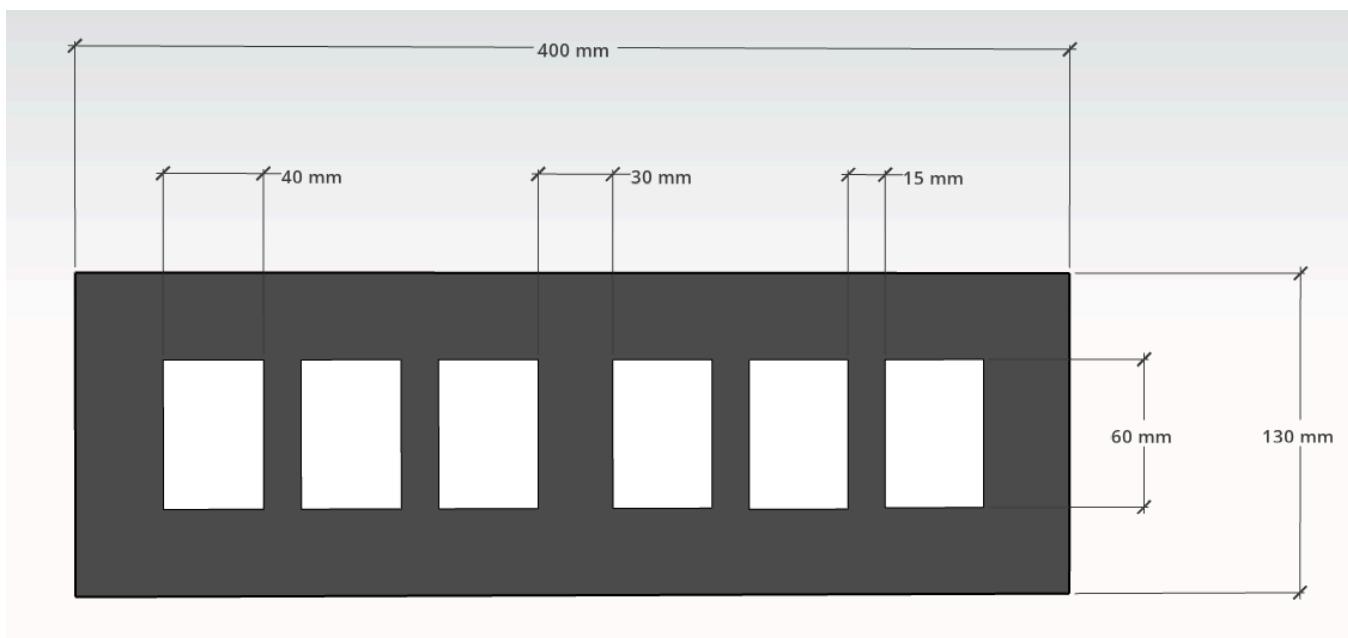
Ancho de los caracteres: 40 mm.

Espesor del trazo: 10 mm.

\*Separación entre caracteres y bloques:

Entre las letras dentro del grupo ("ABC"): Aproximadamente 15 mm.

Entre las letras y los números ("ABC" y "123"): Aproximadamente 30 mm, mayor que la separación entre caracteres individuales para distinguir los bloques fácilmente.



Con esta información, se busca detectar en la imagen, una secuencia de dos grupos de tres componentes, cada uno, mediante esta función:

El patrón esperado es una secuencia de 6 caracteres organizados en dos grupos de 3 caracteres cada uno.

A partir de la imagen filtrada, se vuelven a identificar los componentes conectados (en lugar de pasarlos como parámetros) para tener en los resultados de la función (`num_labels`, `labels`, `stats`, `centroids`) sólo con los componentes de interés.

Los componentes se ordenan en base a su coordenada horizontal (X)

Se calcula la altura promedio de todos los componentes (los posibles caracteres de una patente, debido a sus dimensiones). Si bien en la imagen existen otros componentes que no son los caracteres de interés, sus dimensiones ya se acotaron en la etapa de filtrado, por lo que no afectan demasiado a la altura promedio. De todos modos, se deja un margen en la proporción elegida para los cálculos.

Este cálculo permite asignar dinámicamente, en función de la relación de aspecto que tiene una patente real, los siguientes valores:

- `max_distance_y`: máxima altura en la que un carácter puede diferir del resto de su grupo.
- `e_min`: distancia mínima de espaciado entre los dos grupos de tres caracteres.
- `e_max`: distancia máxima de espaciado entre los dos grupos de tres caracteres.
- `c_min = e_min/2`: distancia mínima de espaciado entre los caracteres de un mismo grupo.
- `c_max = e_min/2`: distancia máxima de espaciado entre los caracteres de un mismo grupo.
- Se busca formar grupos de componentes, controlando que la coordenada Y de los centroides del grupo estén dentro del mismo rango. Las alturas de entre todos los miembros de un grupo no pueden diferir en más que `max_distance_y`. (Ver Observaciones)

- En los subgrupos que cumplan con esta condición y además tengan 6 o más elementos, se van tomando iterativamente, conjuntos de 6 componentes y se verifica que las distancias entre los caracteres de un mismo grupo y la distancia entre los dos grupos, estén dentro de los límites calculados dinámicamente.
- Si se encuentra el patrón XXX XXX, se calculan las coordenadas de la patente:
  - x\_start: coordenada X del primer carácter.
  - x\_end: coordenada X del último carácter más su ancho.
  - y\_start: la mínima coordenada Y entre todos los caracteres.
  - y\_end: la máxima coordenada Y entre todos los caracteres más su altura.
- Se devuelve las coordenadas de la patente y los caracteres detectados.

Si por ejemplo, el grupo tiene 10 componentes, se arman los siguientes subgrupos en 5 iteraciones ( $\text{len}(\text{posibles caracteres}) - 5$ ) como máximo:

iteración 0: 012345

iteración 1: 123456

iteración 2: 234567

iteración 3: 345678

iteración 4: 456789

Así se contemplan todas las secuencias posibles y en la iteración que encuentra el patrón, se termina el proceso.

Detalle para las 12 imágenes de:

- Distancias entre los componentes de un grupo XXX XXX.
- Altura de los componentes de un grupo XXX XXX.
- Umbral de detección.

#### [img01.png](#)

[np.int32(9), np.int32(10), np.int32(18), np.int32(10), np.int32(9)]

[np.int32(152), np.int32(150), np.int32(149), np.int32(147), np.int32(146), np.int32(144)]

110

#### [img02.png](#)

[np.int32(9), np.int32(8), np.int32(17), np.int32(9), np.int32(8)]

[np.int32(210), np.int32(209), np.int32(207), np.int32(204), np.int32(203), np.int32(202)]

110

#### [img03.png](#)

[np.int32(10), np.int32(10), np.int32(21), np.int32(9), np.int32(9)]

[np.int32(193), np.int32(191), np.int32(189), np.int32(185), np.int32(183), np.int32(180)]

110

#### [img04.png](#)

[np.int32(10), np.int32(10), np.int32(19), np.int32(10), np.int32(10)]  
 [np.int32(196), np.int32(192), np.int32(190), np.int32(186), np.int32(184), np.int32(181)]  
 110

**img05.png**

[np.int32(12), np.int32(9), np.int32(21), np.int32(12), np.int32(10)]  
 [np.int32(126), np.int32(124), np.int32(122), np.int32(119), np.int32(117), np.int32(115)]  
 110

**img06.png**

[np.int32(13), np.int32(13), np.int32(26), np.int32(13), np.int32(13)]  
 [np.int32(193), np.int32(190), np.int32(188), np.int32(185), np.int32(184), np.int32(182)]  
 110

**img07.png**

[np.int32(8), np.int32(9), np.int32(16), np.int32(8), np.int32(8)]  
 [np.int32(169), np.int32(169), np.int32(168), np.int32(168), np.int32(168), np.int32(168)]  
 110

**img08.png**

[np.int32(9), np.int32(10), np.int32(18), np.int32(9), np.int32(9)]  
 [np.int32(216), np.int32(214), np.int32(213), np.int32(210), np.int32(209), np.int32(208)]  
 124

**img09.png**

[np.int32(11), np.int32(10), np.int32(22), np.int32(10), np.int32(11)]  
 [np.int32(241), np.int32(240), np.int32(240), np.int32(239), np.int32(239), np.int32(239)]  
 110

**img10.png**

[np.int32(10), np.int32(11), np.int32(18), np.int32(10), np.int32(10)]  
 [np.int32(263), np.int32(263), np.int32(262), np.int32(261), np.int32(260), np.int32(259)]  
 112

**img11.png**

[np.int32(8), np.int32(9), np.int32(17), np.int32(9), np.int32(9)]  
 [np.int32(218), np.int32(217), np.int32(215), np.int32(213), np.int32(211), np.int32(210)]  
 134

**img12.png**

[np.int32(7), np.int32(6), np.int32(13), np.int32(7), np.int32(6)]  
 [np.int32(199), np.int32(199), np.int32(198), np.int32(197), np.int32(196), np.int32(196)]  
 110

**Observaciones:**

Se detectan **errores** cuando en la imagen filtrada existen componentes, cuya coordenada X se encuentra dentro del rango de las coordenadas de la patente, como se muestra en la imagen.

En los grupos de 6 componentes, que busca el código, se filtra un componente que no corresponde.

1

```
[[(np.int32(192), np.int32(216), np.int32(8), np.int32(14), array([197.      , 223.86046512])),  
(np.int32(201), np.int32(214), np.int32(9), np.int32(15), array([204.75675676, 220.62162162])),  
(np.int32(206), np.int32(164), np.int32(7), np.int32(11), array([208.60606061, 168.57575758])),  
(np.int32(211), np.int32(213), np.int32(8), np.int32(15), array([214.5      , 219.71428571])),  
(np.int32(229), np.int32(210), np.int32(8), np.int32(15), array([233.51724138, 217.06896552])),  
(np.int32(238), np.int32(209), np.int32(8), np.int32(15), array([242.66071429, 215.57142857]))]
```

2

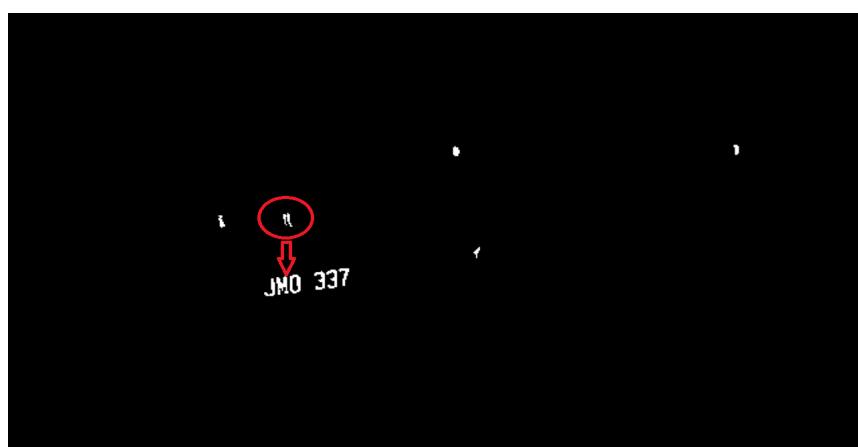
```
[[(np.int32(201), np.int32(214), np.int32(9), np.int32(15), array([204.75675676, 220.62162162])),  
(np.int32(206), np.int32(164), np.int32(7), np.int32(11), array([208.60606061, 168.57575758])),  
(np.int32(211), np.int32(213), np.int32(8), np.int32(15), array([214.5      , 219.71428571])),  
(np.int32(229), np.int32(210), np.int32(8), np.int32(15), array([233.51724138, 217.06896552])),  
(np.int32(238), np.int32(209), np.int32(8), np.int32(15), array([242.66071429, 215.57142857])),  
(np.int32(247), np.int32(208), np.int32(8), np.int32(14), array([251.18604651, 212.86046512]))]
```

3

```
[(np.int32(206), np.int32(164), np.int32(7), np.int32(11), array([208.60606061, 168.57575758])),  
(np.int32(211), np.int32(213), np.int32(8), np.int32(15), array([214.5      , 219.71428571])),  
(np.int32(229), np.int32(210), np.int32(8), np.int32(15), array([233.51724138, 217.06896552])),  
(np.int32(238), np.int32(209), np.int32(8), np.int32(15), array([242.66071429, 215.57142857])),  
(np.int32(247), np.int32(208), np.int32(8), np.int32(14), array([251.18604651, 212.86046512])),  
(np.int32(332), np.int32(111), np.int32(5), np.int32(8), array([333.77419355, 114.77419355]))]
```

4

```
[(np.int32(211), np.int32(213), np.int32(8), np.int32(15), array([214.5      , 219.71428571])),  
(np.int32(229), np.int32(210), np.int32(8), np.int32(15), array([233.51724138, 217.06896552])),  
(np.int32(238), np.int32(209), np.int32(8), np.int32(15), array([242.66071429, 215.57142857])),  
(np.int32(247), np.int32(208), np.int32(8), np.int32(14), array([251.18604651, 212.86046512])),  
(np.int32(332), np.int32(111), np.int32(5), np.int32(8), array([333.77419355, 114.77419355])),  
(np.int32(540), np.int32(110), np.int32(4), np.int32(8), array([541.75, 113.25]))]
```



Para solucionar este problema, previo a la búsqueda de la secuencia, se filtran todos los componentes cuyo centroide tenga la coordenada Y a una distancia mayor a `max_distance_y` que el resto de los componentes del grupo.

```
[(np.int32(201), np.int32(214), np.int32(9), np.int32(15), array([204.75675676, 220.62162162])),  
(np.int32(211), np.int32(213), np.int32(8), np.int32(15), array([214.5      , 219.71428571])),  
(np.int32(229), np.int32(210), np.int32(8), np.int32(15), array([233.51724138, 217.06896552])),  
(np.int32(238), np.int32(209), np.int32(8), np.int32(15), array([242.66071429, 215.57142857])),  
(np.int32(247), np.int32(208), np.int32(8), np.int32(14), array([251.18604651, 212.86046512])),  
(np.int32(332), np.int32(111), np.int32(5), np.int32(8), array([333.77419355, 114.77419355]))]
```

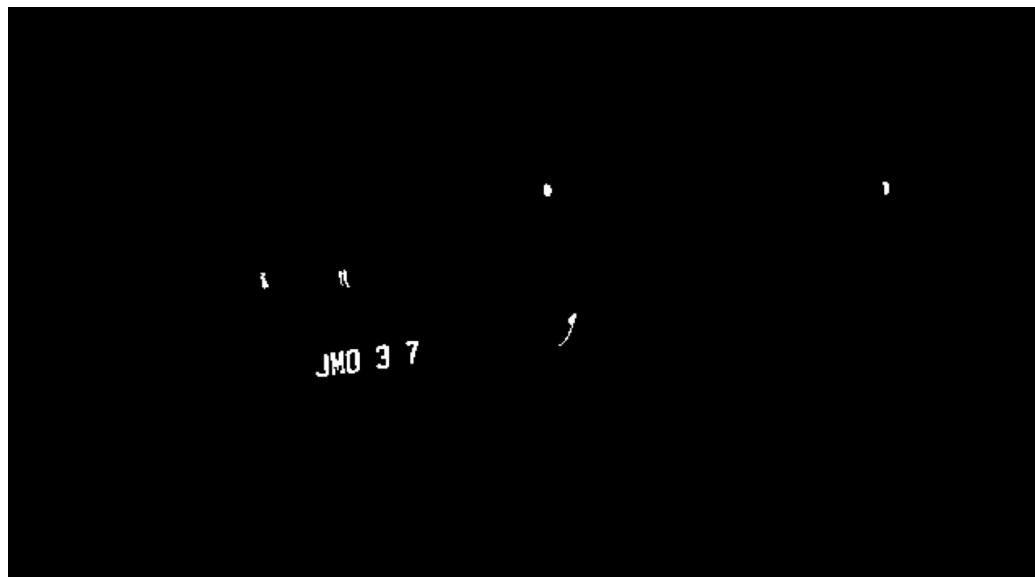
#### **4 -Integración de procesos**

```
def identify_patente(img: np.ndarray)-> dict:
```

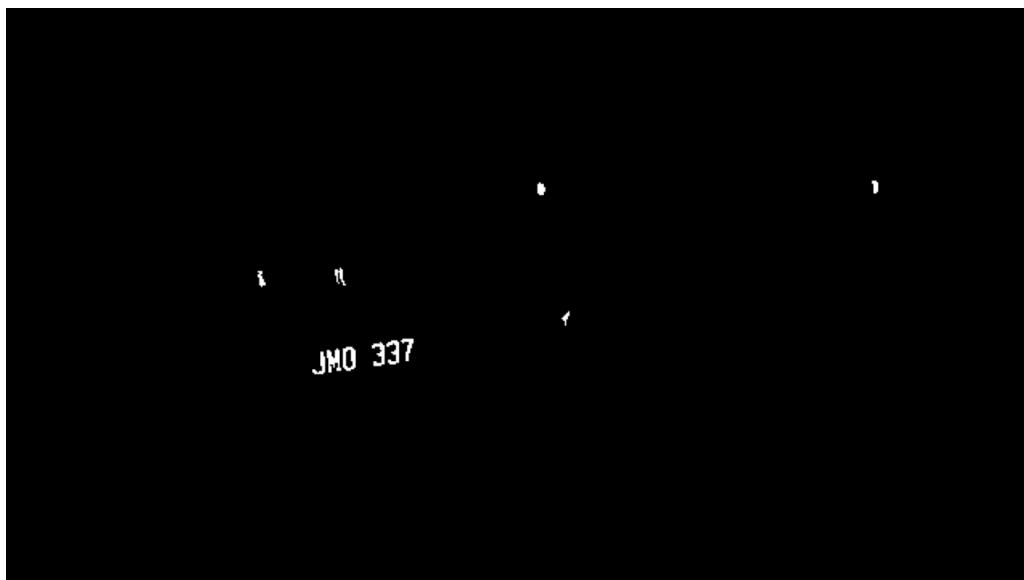
Esta función realiza el proceso iterativo que prueba distintos umbrales de binarización para encontrar la patente en la imagen.

Como los valores para lograr un umbralizado apropiado varían en cada imagen, se itera en un rango de valores determinado empíricamente: (110, 135, 2). Se agrega un intervalo de 2 para reducir las iteraciones a la mitad.

Utiliza todas las funciones previas para procesar, filtrar y encontrar el patrón de patente en la imagen.



Cuando se encuentra un umbral que permite la detección de la secuencia XXX XXX, se sale del bucle y los resultados se almacenan en un diccionario.



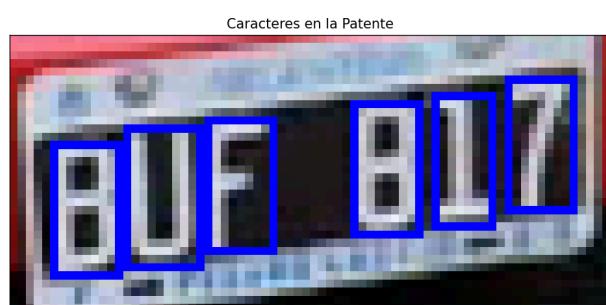
## 5 -Resultados

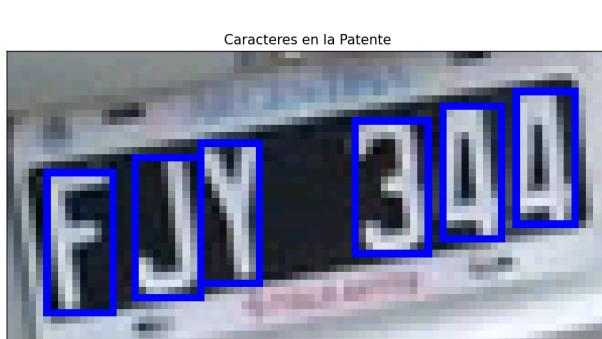
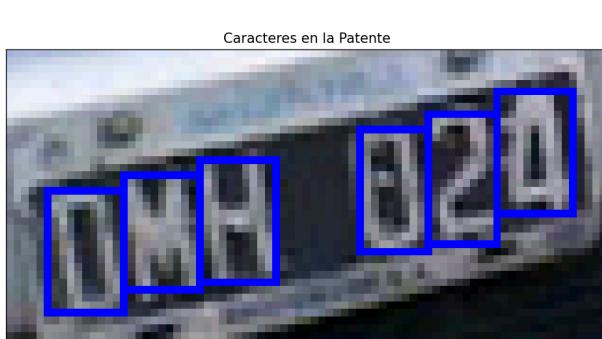
```
def show_results(img: np.ndarray, patente: dict, margin: int)-> None:
```

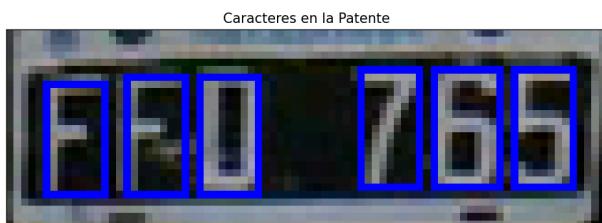
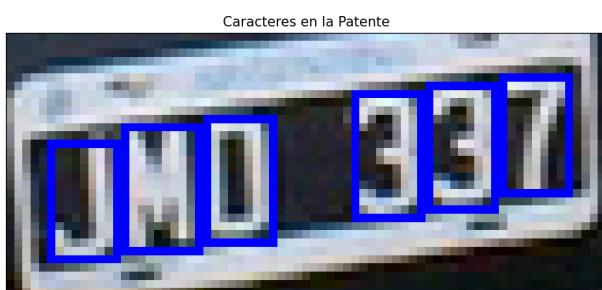
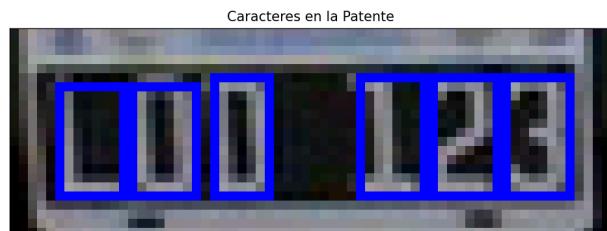
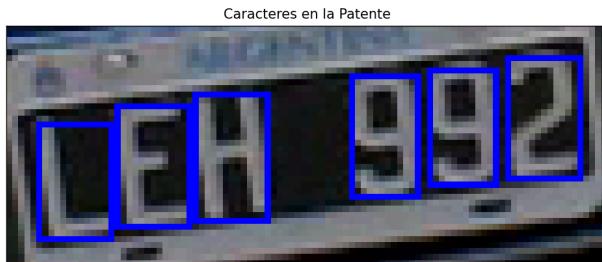
La función:

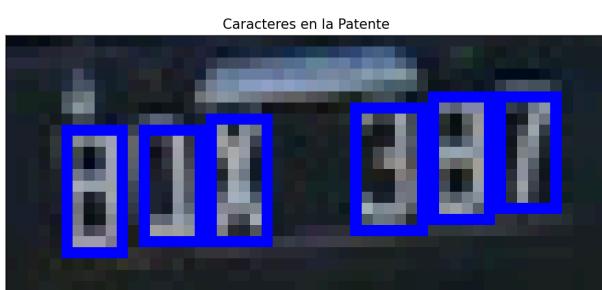
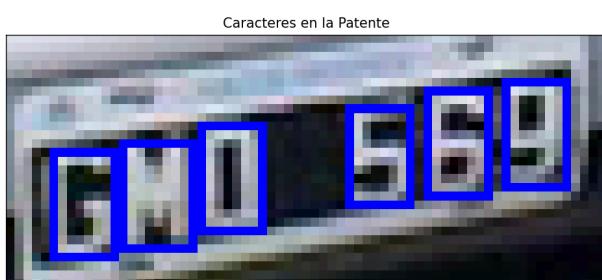
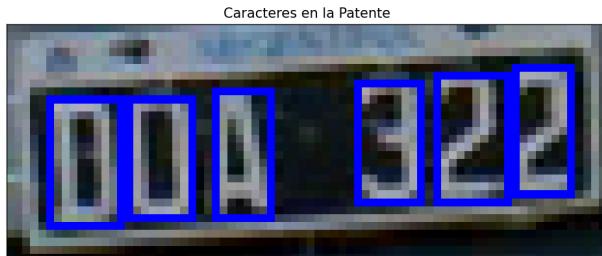
- Dibuja un marco alrededor de la patente detectada, considerando un margen adicional (variable por parámetro).
- Recorta la zona de la patente para mostrar únicamente la región de interés.
- Muestra la segmentación de los caracteres dentro de la patente.

Se identifican las patentes en las 12 imágenes suministradas. Se muestran algunos de los resultados:









## - Repositorio

<https://github.com/Facunditos/PDI-TUIA-TP2-LopezCrespo-Flaibani-Dito.git>