Trabajo Práctico 2

Lucía Masciangelo, Francisco Rodríguez y Barros, Julieta Texier

11-06-2024

Procesamiento de Imágenes I

Tecnicatura en Inteligencia Artificial

Requisitos previos para correr los ejercicios:

1. Clonar el repositorio https://github.com/franciscoryb1/PDI-TUIA-RodriguezYBarros-Texier-Masciangelo
2. Acceder a la carpeta: cd .\TP2
3. Tener python instalado en el entorno a utilizar. Se puede descargar desde https://www.python.org/downloads/
4. Instalar las librerías que serán usadas en los ejercicios:
    - OpenCV: pip install opencv-python
    - Matplotlib: pip install matplotlib
    - numpy: pip install numpy

Ejercicio 1: Detección y clasificación de componentes electrónicos

Primero importamos las librerías y creamos una función para mostrar las imágenes (**imshow**).

Cargamos la imagen 'Placa.png' que se encuentra en la misma carpeta donde va a correr el script. Seguimos los siguientes pasos:

- Convertimos la imagen a escala de grises,
- Aplicamos el filtro de suavizado **MedianBlur** y
- Probamos con varios umbrales de **Canny** para detectar bordes. (elegimos los umbrales 0.2*255 y 0.8*255 para encontrar el chip y los capacitores).
- Después aplicamos Gradiente morfológico con un kernel de (5,5).

Obtuvimos los siguientes resultados:

Imagen con suavizado

Bordes Detectados

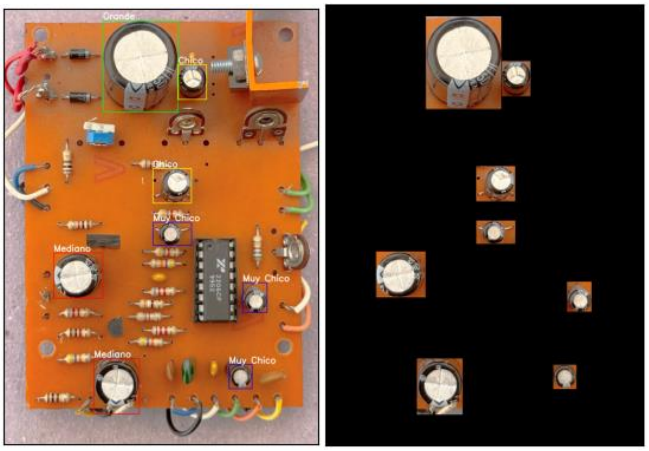
Gradiente morfológico

Después calculamos los componentes conectados y generamos un bounding box para encontrar el chip y otros boundings box para cada uno de los capacitores.

Una vez que tenemos detectado el chip, lo guardamos en una imagen nueva llamada chip.png.

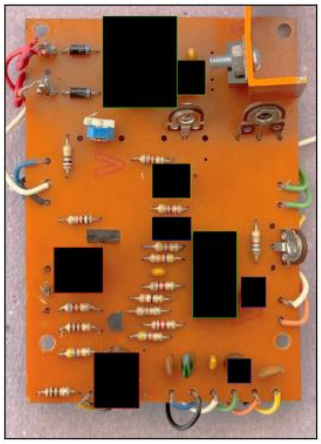
Hacemos lo mismo con los capacitores, pero sumándole una etiqueta que identifique que tamaño tiene el capacitor (también guardamos una imagen llamada capacitores según tamaños, y otra imagen que sea solo los capacitores sin distinción de tamaño)

Además, contamos la cantidad de capacitores por tamaño, quedándonos como resultado lo siguiente:



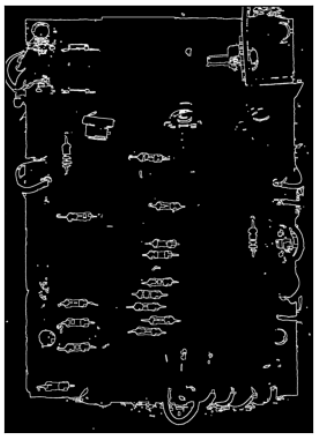

```
>>> print(f"Cantidad de capacitores muy pequeños: {capacitores_muy_chicos}")
Cantidad de capacitores muy pequeños: 3
>>> print(f"Cantidad de capacitores pequeños: {capacitores_chicos}")
Cantidad de capacitores pequeños: 2
>>> print(f"Cantidad de capacitores medianos: {capacitores_medianos}")
Cantidad de capacitores medianos: 2
>>> print(f"Cantidad de capacitores grandes: {capacitores_grandes}")
Cantidad de capacitores grandes: 1
>>> print(f"Cantidad total de capacitores: {capacitores_chicos + capacitores_grandes + capacitores_medianos + capacitores_muy_chicos}")
Cantidad total de capacitores: 8
```

Ahora, para poder detectar las resistencias sin problemas vamos a “eliminar” de la imagen original a los capacitores y al chip, para eso creamos una imagen negra y con componentes conectados detectamos la parte de la imagen donde se encuentran el chip y los capacitores, y después pintamos de la imagen original de negro esos pixeles



Con esta imagen obtenida volvemos a hacer el proceso de:

- Convertir la imagen a escala de grises,
- Binarizamos la imagen con un umbral de 130,
- Aplicamos el filtro de suavizado **MedianBlur** y
- Probamos con varios umbrales de **Canny** para detectar bordes (elegimos los umbrales 0.2*255 y 0.6*255 para encontrar las resistencias)
- Aplicamos clausura con el kernel (20,1) y
- Por último, aplicamos **Gradiente morfológico** con un kernel de (5,5).

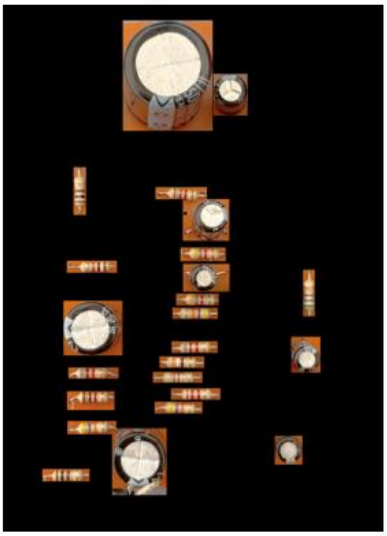


Después calculamos los componentes conectados y aplicamos bounding box para encontrar las resistencias.

Por último, agregamos las resistencias a la imagen con fondo negro en la que ya tenemos el chip y los capacitores y creamos otra solo con las resistencias, en ese mismo código pusimos un contador para después imprimir por consola la cantidad de resistencias encontradas.

En resumen, la salida del ejercicio es:

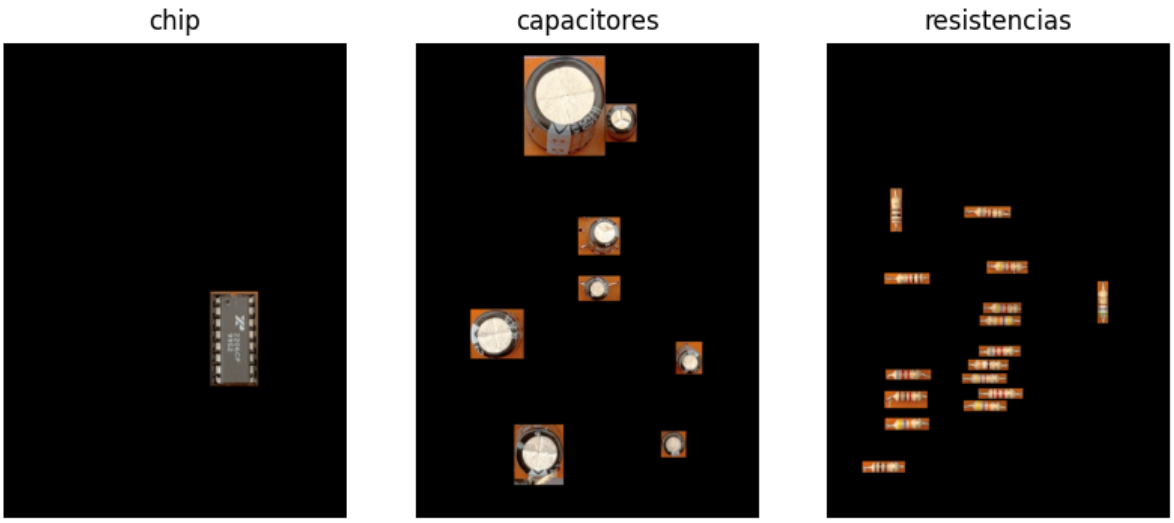
todas las componentes detectadas



capacitores con sus tamaños


```
>>> print(f"Cantidad de capacitores muy pequeños: {capacitores_muy_chicos}")
Cantidad de capacitores muy pequeños: 3
>>> print(f"Cantidad de capacitores pequeños: {capacitores_chicos}")
Cantidad de capacitores pequeños: 2
>>> print(f"Cantidad de capacitores medianos: {capacitores_medianos}")
Cantidad de capacitores medianos: 2
>>> print(f"Cantidad de capacitores grandes: {capacitores_grandes}")
Cantidad de capacitores grandes: 1
>>> print(f"Cantidad total de capacitores: {capacitores_chicos + capacitores_grandes + capacitores_medianos + capacitores_muy_chicos}")
Cantidad total de capacitores: 8
```

```
>>> print(f"Cantidad de resistencias eléctricas: {contador}")
Cantidad de resistencias eléctricas: 16
```



Ejercicio 2: Detección de patentes

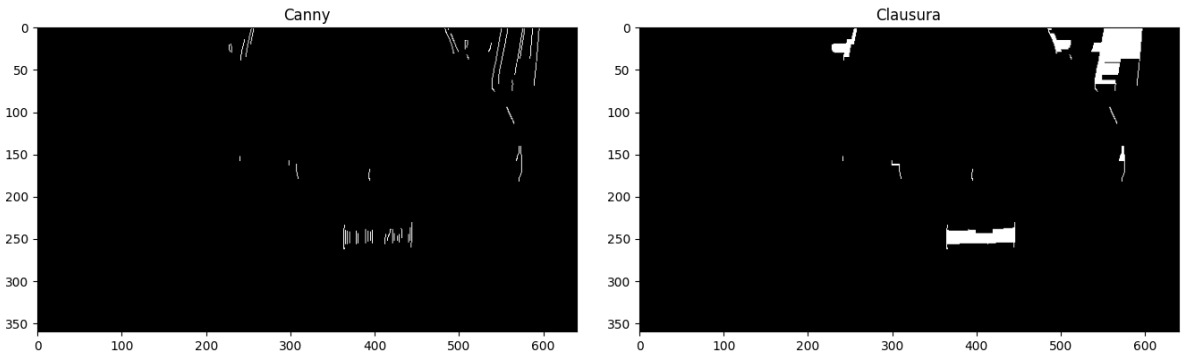
Al igual que en el ejercicio 1 importamos las librerías necesarias y definimos la función **imshow** para mostrar las imágenes.

Contamos con una carpeta, donde se encuentran 12 imágenes de autos, dónde el objetivo es detectar la patente y sus componentes (letras y números).

En primer lugar buscamos encontrar y recortar la patente correspondiente de cada imagen, para ello definimos la función **recortar_patente(**img**)** la cual acepta como entrada una imagen sin procesar y devuelve una imagen correspondiente al recorte de la patente.

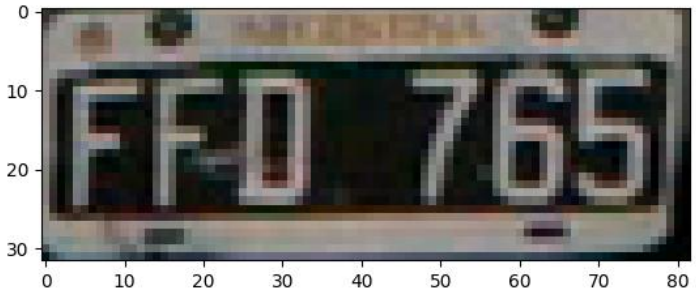
Dentro de esta función seguimos varios pasos para lograr el resultado final:

- Convertimos la imagen a escala de grises.
- Aplicamos un filtro Gaussiano **GaussianBlur**.
- Usamos **Canny** y aplicamos técnicas de morfología, **Clausura** con el objetivo de cerrar los elementos y que quedaran de forma uniforme, para luego poder detectar componentes conectados.



Nota: en esta función probamos haciendo umbralado, y otros métodos de morfología, pero no obtuvimos buenos resultados.

- Encontramos los componentes 8 conectados de la imagen y los filtramos por dos condiciones, que el **Área** sea mayor a 300 px y que el ancho sea mayor al alto, teniendo en cuenta la relación de aspecto de la patente.
- Finalmente, a partir del componente encontrado, obtuvimos las coordenadas haciendo uso de los valores de **Stats**, realizamos el recorte correspondiente sobre la imagen de entrada original y retornamos la patente. Mostramos un ejemplo:

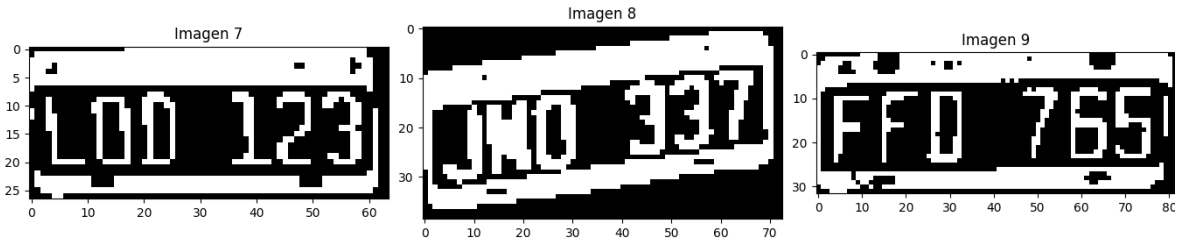


El paso final hacia el resultado objetivo lo desarrollamos con la función **detectar_componentes(**img**)** esta función al igual que la anterior, toma una imagen como entrada y devuelve una imagen resultante con la patente recortada y sus caracteres detectados.

A la imagen resultante de la función **recortar_patente(**img**)** le aplicamos los siguientes pasos:

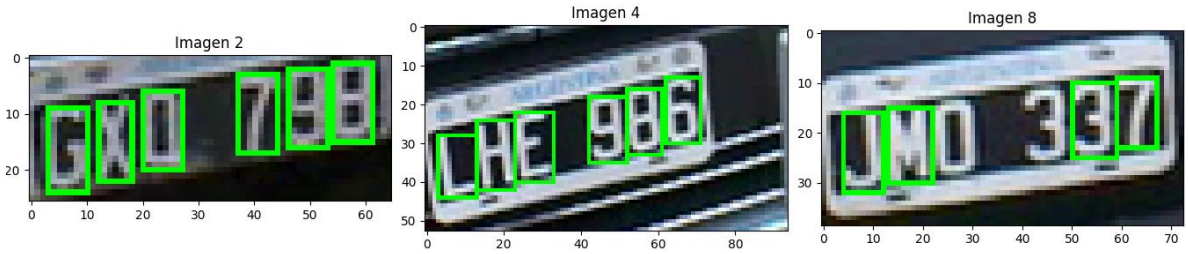
- **Binarizar** la imagen, hicimos pruebas con diferentes umbrales analizando la pérdida y ganancia en cada foto con cada umbral y decidimos finalmente quedarnos con un umbral de 113 ya que obtuvimos los mejores resultados.

Estos son algunos de los resultados obtenidos



- El siguiente paso fue detectar componentes conectados, en este caso también hicimos pruebas con conectividad 4 y conectividad 8, finalmente nos quedamos con 4 debido a que en varios casos teníamos caracteres conectados entre sí o con el borde en forma diagonal. De igual forma hay algunas imágenes (como en la imagen 8 presentada arriba) donde los componentes se conectan de forma vertical con el borde de la patente y esto hace que no detecte correctamente el carácter,

- El paso final es filtrar los componentes encontrados teniendo en cuenta la relación de aspecto entre el alto y ancho de los caracteres, así como también, su área, y dibujar el boundingbox correspondiente a cada carácter.



En conclusión, de las 12 patentes fotografiadas, terminamos encontrando 7 patentes completas, 2 patentes con 5 caracteres encontrados de 6, otras 2 con 4 caracteres encontrados, y 1 sola patente que únicamente encontramos 1 carácter.

Finalmente detectamos 61 caracteres de los 72 totales, lo que representa el 84,72% de acierto.