Universidad Nacional de Rosario

Trabajo práctico Nº 01

Procesamiento de Imágenes

Tecnicatura Universitaria en Inteligencia Artificial

Periodo: Octubre 2025

Integrantes

- de Brito, Nicolas
- Giacone, Agustin
- Taborda, Matias

Índice

Introducción

En el presente informe se muestra cómo se resolvieron los problemas planteados utilizando distintas técnicas aprendidas en la materia. En cada ejercicio se explican los pasos seguidos, los inconvenientes que fueron apareciendo y las soluciones que se aplicaron para resolverlos.

Además, se incluyen imágenes que ayudan a comprender mejor los resultados obtenidos y, en algunos casos, ejemplos prácticos del funcionamiento de los programas. El trabajo se realizó a partir de un repositorio en GitHub que cuenta con un archivo README con las instrucciones para ejecutarlo, dos archivos en Python con las soluciones y los materiales brindados por la cátedra.

Problema 1 - Ecualización local de histograma

Contexto del problema

En este ejercicio se trabajó con una imagen en formato **.tif** que posee un fondo gris uniforme y cinco figuras negras, cada una ocultando un objeto. Para poder visualizar esos elementos, fue necesario aplicar una **ecualización local del histograma**, utilizando una ventana (en nuestro caso, cuadrada) que se desplaza píxel a píxel, centrada en cada punto de la imagen hasta recorrerla por completo.

No se aplicó una ecualización global del histograma, ya que este método no resulta eficaz para resaltar detalles presentes en zonas oscuras, como las que tiene esta imagen.

Resolución del problema

Para abordar este ejercicio se utilizaron las librerías OpenCV (cv2), NumPy (numpy) y Matplotlib (matplotlib.pyplot). La imagen original, en formato .tif, se procesó en escala de grises para simplificar el análisis y el tratamiento posterior.

Se implementó una función encargada de realizar la ecualización local del histograma, siguiendo los pasos que se detallan a continuación:

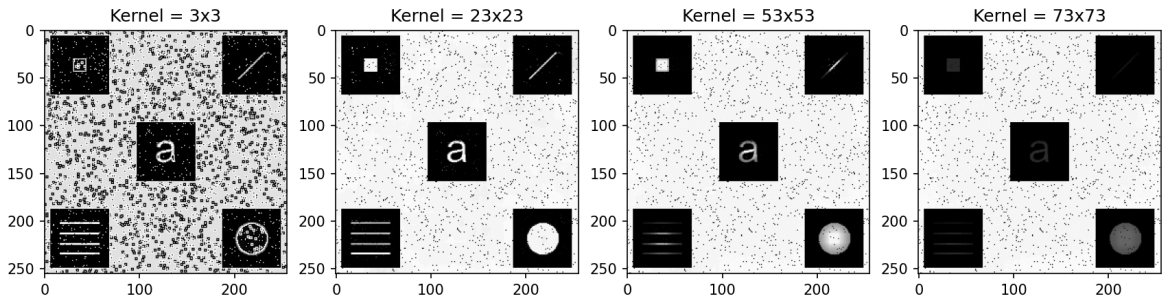
- Se agregaron bordes replicados a la imagen utilizando la función cv2.copyMakeBorder, con el fin de evitar problemas en las zonas próximas a los límites.

- Se creó una imagen vacía del mismo tamaño y tipo de datos que la original, inicializada en negro, destinada a almacenar los resultados de la transformación.

- Luego, la imagen se recorrió píxel por píxel. En cada posición, se extrajo una ventana de tamaño M x N centrada en el píxel actual, sobre la cual se aplicó la ecualización del histograma. Los valores de M y N los fuimos probando de manera experimental hasta encontrar valores que sean acordes.

- Finalmente, el valor ecualizado correspondiente al píxel central de cada ventana se guardó en la imagen resultante.

A continuación se muestra el resultado obtenido al aplicar el método con cuatro tamaños de ventana distintos:



Al comparar los resultados, puede observarse que cuando la ventana es muy pequeña (3x3), la imagen presenta un nivel de ruido elevado, lo que dificulta distinguir de manera completamente limpia las figuras. En cambio, al utilizar ventanas mucho más grandes (53x53 y 73x73), el ruido disminuye, pero los objetos ocultos pierden definición y contraste (a medida que el tamaño de la ventana aumenta, esta condición se agrava).
La opción más equilibrada que encontramos es la ventana intermedia (23x23), que permite una imagen con menor ruido y una mejor visualización de los objetos ocultos, sin pérdida de definición de los mismos.

Problemas durante el desarrollo

Durante la implementación del algoritmo se presentaron algunos inconvenientes que fue necesario resolver para obtener un resultado correcto.

Uno de los principales problemas se dio con el tamaño de la ventana (M x N) utilizada en la ecualización local. Si los valores elegidos son pares, la ventana no posee un punto central exacto, lo que impide ubicar correctamente el píxel que debe recibir el valor ecualizado. Por esa razón, el tamaño de la ventana debe ser impar, garantizando así una región simétrica en torno al píxel central. Por ejemplo, una ventana de 3x3 tiene un centro en la posición (1,1), mientras que una de 4x4 no tiene un punto central único, lo que genera errores en el cálculo.

Otro inconveniente surgió al procesar los bordes de la imagen. Cuando la ventana se ubica cerca de los límites, faltan píxeles alrededor para completar la región de análisis. Para resolver esto, fue necesario agregar bordes artificiales a la imagen, replicando los valores más cercanos mediante la función cv2.copyMakeBorder.

Inicialmente no nos habíamos percatado que, al aumentar los valores de la ventana, la cantidad de píxeles necesarios en los bordes de la imagen original cambiaba. Eso generó que, en un comienzo, la imagen se deformara en alguna de sus partes en vez de ajustar su color. Luego pudimos entender que necesitábamos tener una cantidad de pixeles equivalentes a la mitad del tamaño de nuestra ventana cuadrada

(si nuestra ventana tenía un tamaño de 3x3, necesitábamos tener 3//2 = 1 cantidad de pixeles por sobre el borde original).

Problema 2 - Validación de formularios

Explicación del problema

En este ejercicio se trabajó con un conjunto de formularios digitalizados que presentan un formato estructurado con distintos campos a completar, como Nombre y Apellido, Edad, Mail, Legajo, Preguntas y Comentarios. Cada formulario pertenece a una versión distinta (A, B, C o D), aunque todos comparten el mismo diseño general.

El objetivo fue desarrollar un sistema que pueda leer y analizar las imágenes de estos formularios para verificar de forma automática si fueron completados correctamente según las condiciones definidas para cada campo.

Para eso, el programa debía cumplir con las siguientes consignas:

A. Tomar como entrada únicamente la imagen de un formulario y mostrar por pantalla qué campos fueron completados correctamente (OK) y cuáles no (MAL).

B. Aplicar el algoritmo de manera automática sobre un conjunto de cinco formularios completos (formulario_01.png a formulario_05.png) e informar los resultados según la versión del formulario (A, B, C o D).

C. Generar una única imagen de salida que muestre qué personas completaron correctamente el formulario y cuáles no. En esa imagen deben aparecer los recortes del campo Nombre y Apellido, acompañados de una marca o texto que indique su resultado (OK o MAL).

D. Crear un archivo CSV que almacene los resultados obtenidos de cada validación. En este archivo, cada fila representa un formulario procesado e incluye el ID del formulario y el estado (OK o MAL) de cada uno de sus campos.

Este trabajo combina técnicas de procesamiento digital de imágenes con validación lógica de datos, permitiendo automatizar una tarea que habitualmente se realiza de manera manual. De esta forma, se logra un proceso más rápido, preciso y confiable, reduciendo los posibles errores humanos en la revisión de formularios.

Resolución del problema

1) Carga de imágenes

Empezamos por la lectura de los formularios. Para eso usamos la función `cargar_imagenes(ids)`, que recibe una lista de identificadores (01, 02, …, 05) y carga cada archivo formulario_{id}.png en escala de grises con OpenCV.

Esto nos deja una lista de matrices (una por imagen) y nos permite procesar varios formularios de corrido.

FORMULARIO A		
Nombre y apellido		
Edad		
Mail		
Legajo		
	Si	No
Pregunta 1		
Pregunta 2		
Pregunta 3		
Comentarios		

FORMULARIO A		
Nombre y apellido	JORGE	
Edad	4500	
Mail	JORGE @GMAIL.COM	
Legajo	X45ASLAB W45	
	Si	No
Pregunta 1		
Pregunta 2	X	x
Pregunta 3		xx
Comentarios	ESTE ES UN COMENTARIO MUY MUY LARGO.	

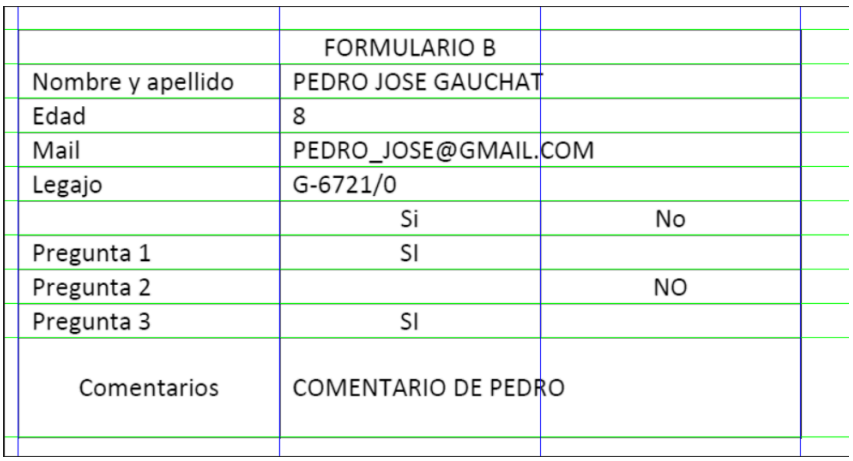
---

2) Detección de líneas y armado de la grilla

Para ubicar los campos, primero binarizamos por umbral (comparando cada píxel con un valor de gris) y calculamos la suma por filas y por columnas. La lógica es que las líneas de la tabla acumulan muchos píxeles oscuros y aparecen como picos en esas proyecciones.

Con eso detectamos las posiciones de las líneas horizontales y verticales y luego agrupamos posiciones cercanas con `agrupar_lineas(posiciones, distancia_minima)` para quedarnos con una sola coordenada por línea (evitando duplicados cuando la línea tiene más de 1 píxel de grosor).

Todo este paso está encapsulado dentro de `validar_imagen(imagen)` (es la función central del proceso). Esta función recibe como parámetro una imagen y devuelve un diccionario con elementos que se describirán posteriormente.



	FORMULARIO B	
Nombre y apellido	PEDRO JOSE GAUCHAT	
Edad	8	
Mail	PEDRO_JOSE@GMAIL.COM	
Legajo	G-6721/0	
	Si	No
Pregunta 1	SI	
Pregunta 2		NO
Pregunta 3	SI	
Comentarios	COMENTARIO DE PEDRO	

---

3) Recortes por campo (sub imágenes)

Con las posiciones de líneas, recortamos las sub imágenes de cada campo:

- Para Nombre, Edad, Mail, Legajo y Comentarios, tomamos el bloque entre dos líneas horizontales y el rango de columnas válido (excluyendo bordes para evitar que molesten). Una particularidad de estos campos es que las restricciones se evalúan sobre una sola imagen, por ende no se utilizan todas las líneas verticales detectadas en los espacios donde se completa el formulario.

- Para Preguntas 1–3, recortamos dos imágenes (Sí/No) ya que las restricciones se evalúan a partir del resultado del procesamiento de ambas, por ende será necesario utilizar las tres últimas líneas verticales detectadas.

Este recorte también se encuentra en el cuerpo de `validar_imagen`.

Luego, se llama a la función cuenta_elementos(campo) para contar la cantidad de caracteres y palabras detectadas en la imagen (se detalla en el punto 4). En el caso de las preguntas, como su evaluación requiere contabilizar elementos de dos imágenes, se itera sobre los dos recortes y se acumula la cantidad de caracteres y palabras para su posterior validación de restricciones.

A continuación, dependiendo el campo a validar, se llama a una función que chequea las restricciones planteadas en base a cantidad de caracteres y cantidad de palabras (se detalla en punto 5).

Finalmente, se devuelve un diccionario que tiene como clave un texto que hace referencia al nombre del campo y, en su valor, una tupla con el estado del campo (OK/MAL) y el crop correspondiente al campo.

---

4) Conteo de caracteres y palabras

Para analizar el contenido, usamos la función `cuenta_elementos(campo)`. La misma recibe el recorte de la imagen original (campo) y realiza las siguientes tareas:

- Umbraliza el recorte para binarizar la imagen y cambiamos tipo de dato para poder utilizar funciones acordes.
- Obtiene componentes conectados con `cv2.connectedComponentsWithStats`.
- Cuenta cuántos caracteres hay (cantidad de componentes) y estima cuántas palabras midiendo separaciones grandes entre componentes consecutivas (si la distancia supera un umbral, lo tomamos como espacio).

La función devuelve dos valores: cantidad de caracteres y cantidad de palabras. Los mismos servirán para simplificar las posteriores validaciones.

---

5) Reglas de validación por tipo de campo

Las reglas están implementadas como funciones simples que reciben dos parámetros (cantidad de caracteres y cantidad de palabras) y devuelven True/False:

- `valida_nombre` → al menos 2 palabras y hasta 25 caracteres.
- `valida_edad` → valida que cantidad de caracteres sea igual a 2 o 3 y cantidad de palabras igual a 1.
- `valida_mail` → 1 palabra y hasta 25 caracteres.
- `valida_legajo` → exactamente 8 caracteres, sin espacios (una palabra).

- `valida_pregunta` → exactamente 1 marca (ni vacía ni doble marca). En otras palabras, valida de la cantidad de caracteres para los dos recortes (Si/No) sea igual a 1 y cantidad de palabras sea igual a 1.

- `valida_comentario` → Más de una palabra y menor o igual a 25 caracteres.

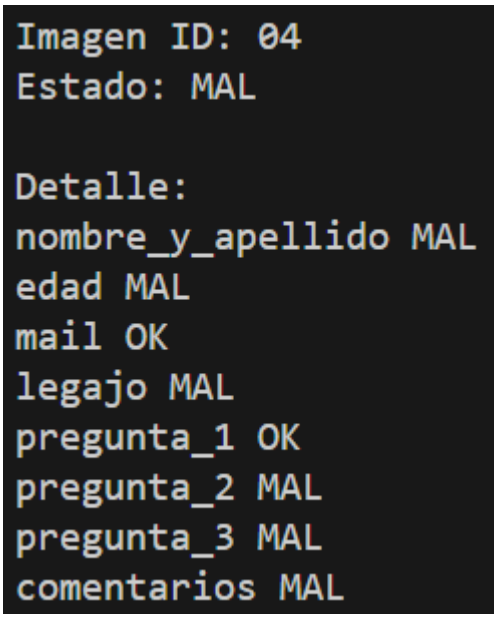
---

6) Procesamiento en lote y salida visual

Como el trabajo pedía poder procesar las imágenes en lotes iterando sobre una función central, se decidió crear una función llamada `generar_validaciones(ids, imagenes)` que recibe como parámetro una lista de id y una lista de imágenes y devuelve un arreglo de diccionarios, además de mostrar por pantalla los estados de los campos.

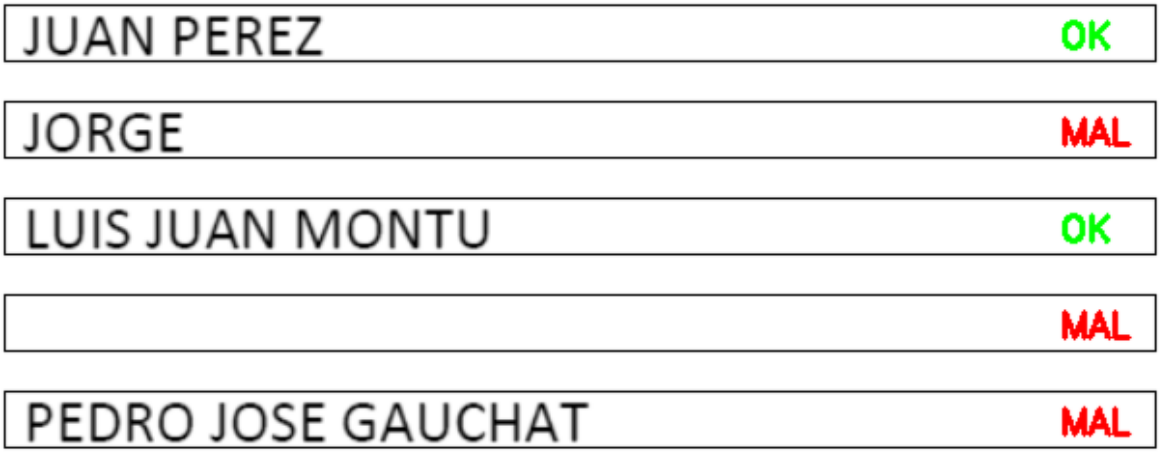
En esencia, esta función llama a `validar_imagen` para cada imagen y muestra por consola el estado de cada campo. A su vez, acumula los diccionarios devueltos por `validar_imagen` y los devuelve en el arreglo de diccionarios mencionados anteriormente.

FORMULARIO B		
Nombre y apellido		
Edad	1 5	
Mail	CASILLAMUYLARGAD@GMAIL.COM	
Legajo	M-98784/1	
	Si	No
Pregunta 1	O	
Pregunta 2	Oo	
Pregunta 3	ooo	
Comentarios		



Luego, para una vista rápida de resultados a nivel de comprobantes, generamos una imagen resumen con `generar_imagen_validaciones(validaciones)`. Esa

función tiene como objetivo generar una imagen que pega el recorte del campo “Nombre y Apellido” de cada formulario y le escribe al lado OK/MAL según el estado general del formulario. Para generar dicho estado de formulario se creó una función llamada `estado_formulario` que recibe como parámetro el diccionario que devuelve `validar_imagen` y simplemente itera sobre los campos chequeando si alguno de ellos se encuentra en estado MAL. De ser así, devuelve MAL, sino devuelve OK. Es una manera de encapsular una pequeña lógica para chequear el estado general del formulario. Finalmente se logra obtener una imagen similar a la que se muestra a continuacion:



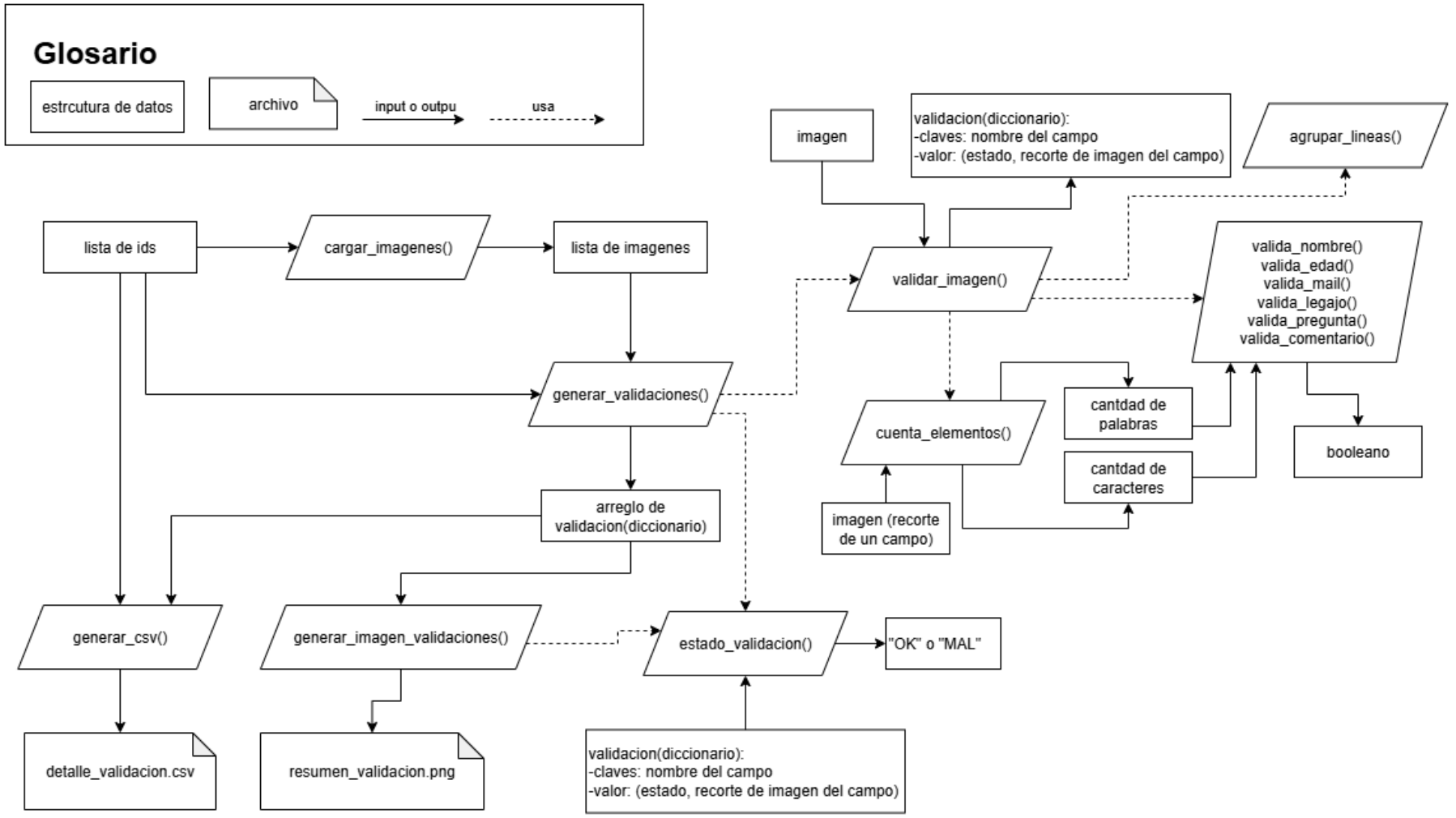
---

7) Registro en CSV

Para generar el archivo CSV, usamos `generar_csv(ids, validaciones)`, que crea el elemento `detalle_validacion.csv` en la carpeta del proyecto. La estructura del archivo cuenta con una fila por formulario y columnas en el orden pedido por la cátedra (ID, nombre_y_apellido, edad, mail, legajo, pregunta_1, pregunta_2, pregunta_3, comentarios), cada una con OK/MAL.
Esto facilita entregar evidencia y volver a revisar resultados sin tener que abrir las imágenes.

ID	nombre_y_apellido	edad	mail	legajo	pregunta_1	pregunta_2	pregunta_3	comentarios
01	OK	OK	OK	OK	OK	OK	OK	OK
02	MAL	MAL	MAL	MAL	MAL	MAL	MAL	MAL
03	OK	OK	OK	OK	OK	OK	OK	OK
04	MAL	MAL	OK	MAL	OK	MAL	MAL	MAL
05	OK	MAL	OK	OK	MAL	MAL	MAL	OK

Mapa de funciones del código



Conclusión

A lo largo de este trabajo pudimos aplicar distintos conceptos de procesamiento de imágenes para resolver dos problemas diferentes. En el primer ejercicio, utilizamos la ecualización local del histograma para resaltar detalles ocultos dentro de una imagen, analizando cómo influye el tamaño de la ventana en la calidad del resultado. Esto nos permitió comprender mejor la importancia de elegir correctamente los parámetros según las características de la imagen.

En el segundo ejercicio, desarrollamos un sistema capaz de validar formularios de manera automática a partir de imágenes, detectando cada campo y verificando si su contenido cumplía con las condiciones establecidas. Gracias a las distintas funciones implementadas, logramos automatizar un proceso que normalmente se haría de forma manual, reduciendo el margen de error y obteniendo resultados precisos y organizados tanto en consola como en formato visual y en archivo CSV.

En general, el trabajo nos permitió integrar herramientas de OpenCV y NumPy con la lógica de validación, reforzando el vínculo entre la programación y el análisis de imágenes. Además, nos ayudó a entender cómo dividir un problema complejo en etapas más simples y cómo verificar los resultados de forma clara y visual.