Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Tecnicatura Universitaria en Inteligencia Artificial

PROCESAMIENTO DE IMÁGENES 1

Informe:
“Trabajo Práctico 1”

Fecha: 21/10/2025

Autores:

- CORRADINI, JULIO - Legajo: C-7525/6
- LEÓN, ESTEBAN - Legajo: L-3413/4
- SAAD, JUAN MANUEL - Legajo: S-5803/3
- ZIMMER, FEDERICO - Legajo:Z-1239/4

Docentes:

- SAD, GONZALO
- CALLE, JUAN MANUEL
- ALLIONE, JOAQUIN

2025

Problema 1 - Ecualización Local del Histograma

Problemática

El desafío de este problema es desarrollar e implementar la técnica de Ecualización Local del Histograma (LHE) para mejorar el contraste en regiones específicas de una imagen que contienen detalles de bajo contraste, donde una ecualización global sería inefectiva debido a que la distribución de intensidades global enmascara las diferencias locales.

Puntos a Desarrollar

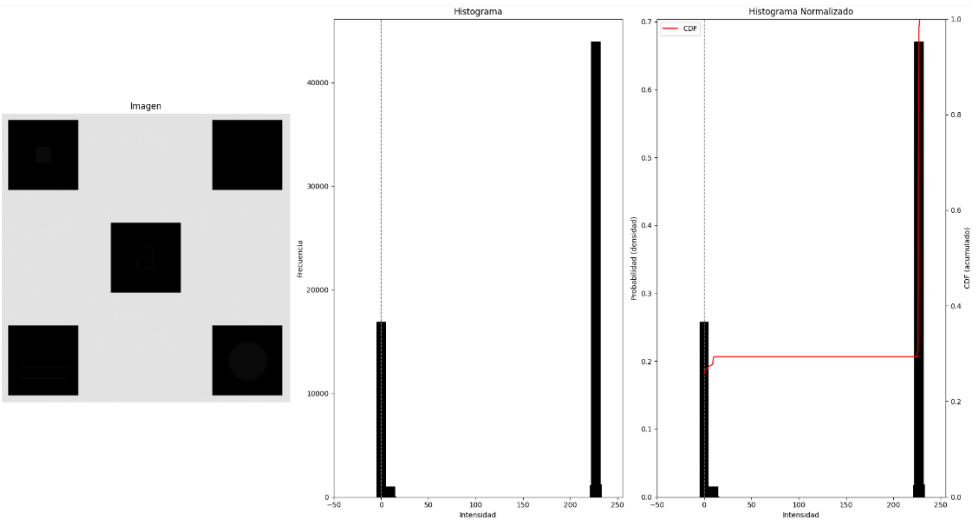
El usuario debe completar las siguientes tareas:

1. **Desarrollo de la Función LHE:**
    - Desarrollar una función que implemente el algoritmo de Ecualización global del Histograma.
    - La función debe recibir como parámetros de entrada:
        - La Imagen a procesar.
        - El tamaño de la ventana de procesamiento.
2. **Aplicación y Revelación de Detalles:**
    - Aplicar la función a la imagen de ejemplo.
    - Informar cuáles son los detalles escondidos que se revelan en las diferentes zonas de la imagen procesada.
3. **Análisis de la Ventana:**
    - Realizar un análisis sobre cómo influye el tamaño de la ventana en los resultados obtenidos, comentando los efectos visuales y la naturaleza de la mejora de contraste para diferentes tamaños.

Resolución

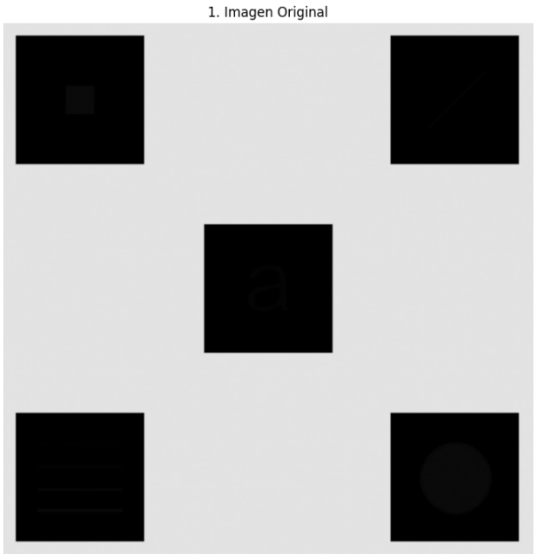
Lo primero que realizamos fue un análisis de las distintas tonalidades de gris presentes en la imagen original.

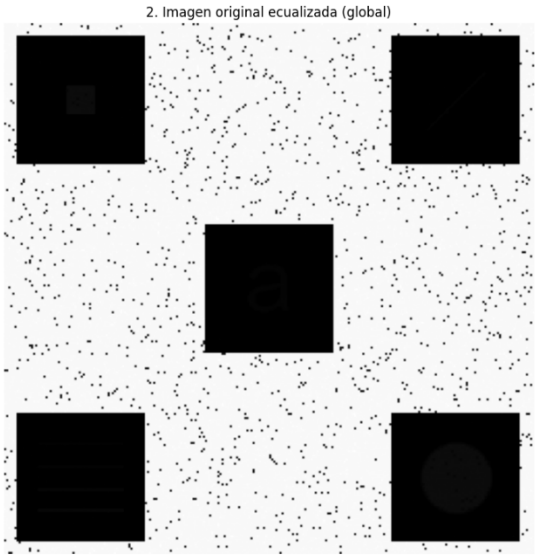
Para ello, graficamos el histograma y el histograma normalizado, lo que permitió visualizar la distribución de los valores de intensidad a lo largo del rango dinámico.



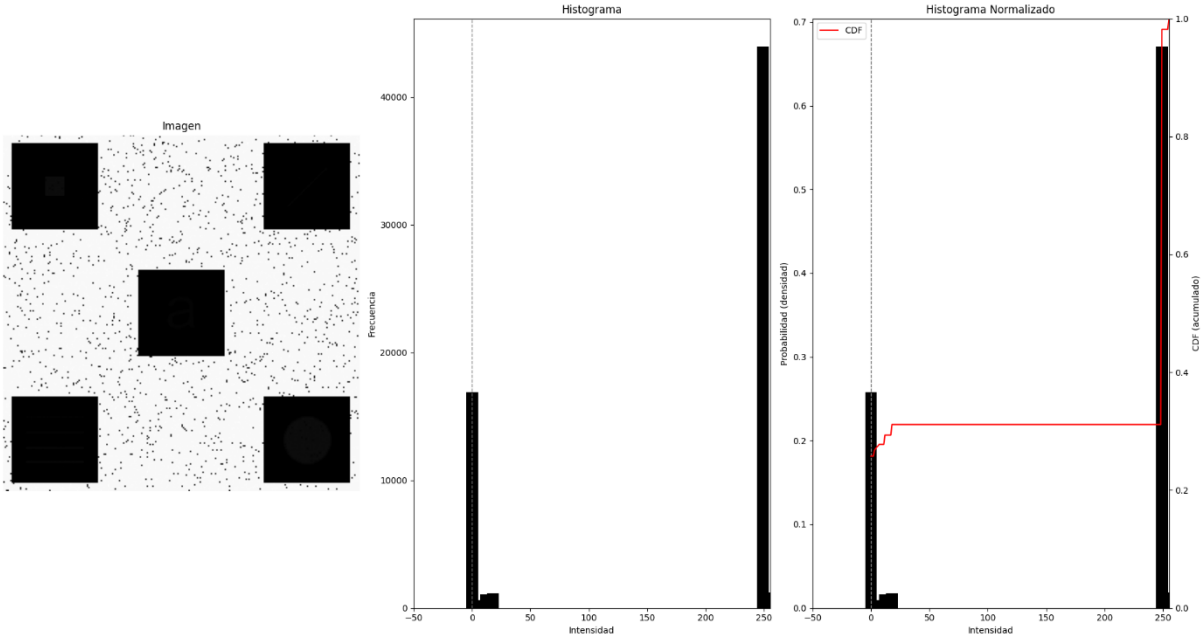
A partir de este análisis observamos que existen zonas con niveles de gris muy próximos entre sí, lo que genera un bajo contraste y hace que algunos detalles sean difícilmente perceptibles a simple vista.

A continuación, aplicamos una ecualización global del histograma para mejorar el contraste general de la imagen.



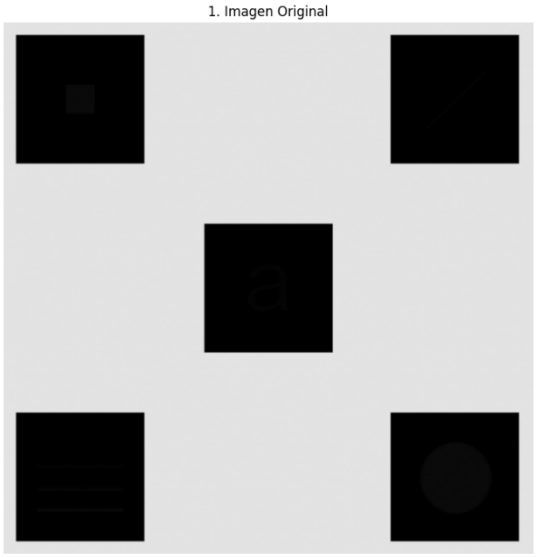


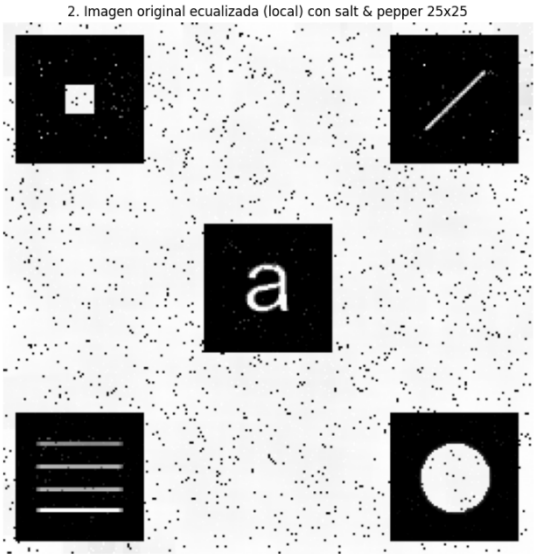
Sin embargo, comprobamos que este método no resulta suficiente para resaltar las regiones de interés, ya que los detalles ocultos continúan poco visibles.
Esto se debe a que la imagen original presenta sectores con contrastes elevados de forma natural, lo que provoca que la ecualización global redistribuya las intensidades de forma poco uniforme.



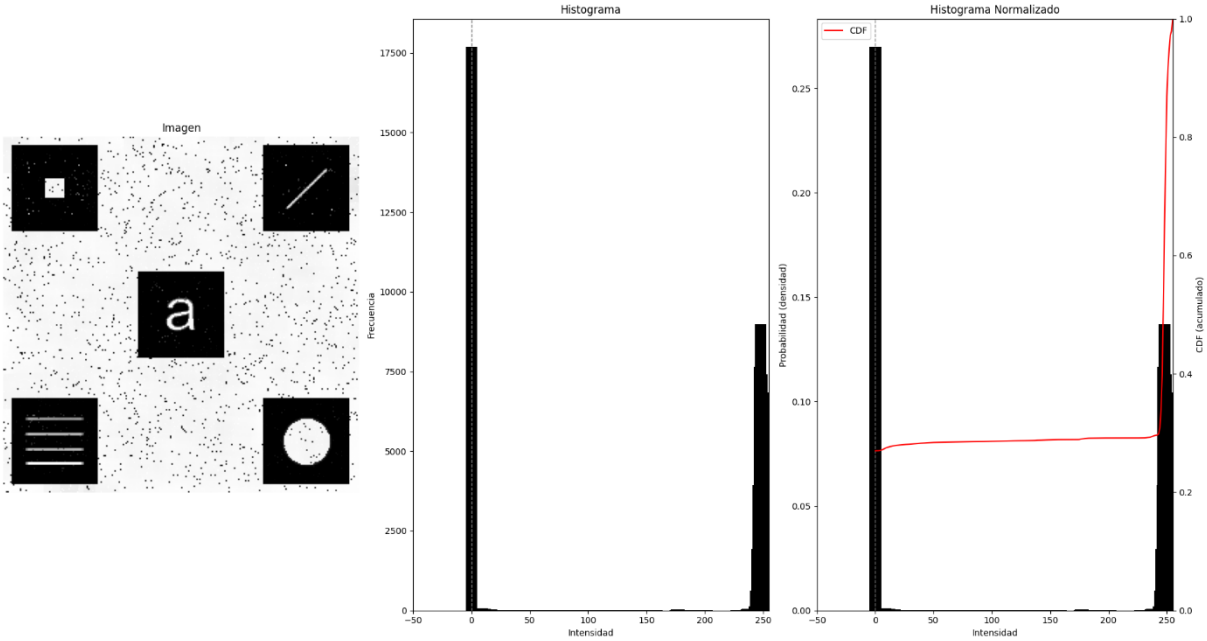
Con el objetivo de mejorar los resultados, implementamos una ecualización local mediante una ventana deslizante, cuyo centro se desplaza píxel a píxel sobre toda la imagen.
Este método permitió ajustar el contraste de manera localizada según las particularidades de cada zona.
El resultado mostró una mejor definición en las zonas de bajo contraste, permitiendo distinguir claramente las figuras ocultas en los distintos sectores del recuadro:

1. **Esquina superior izquierda:** un cuadrado pequeño centrado.
2. **Esquina superior derecha:** una barra diagonal.
3. **Centro:** la letra “a”.
4. **Esquina inferior izquierda:** cuatro líneas horizontales paralelas.
5. **Esquina inferior derecha:** un círculo.

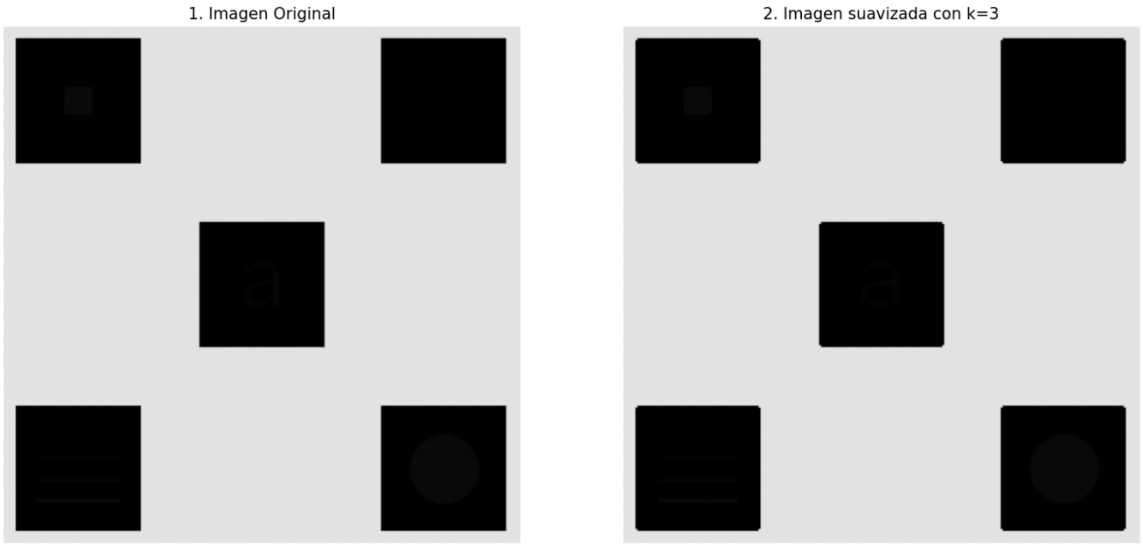




No obstante, la ecualización local incrementó el ruido tipo “salt and pepper”, originado por valores de píxeles extremadamente altos o bajos.



Para mitigar este problema, decidimos preprocesar la imagen aplicando un filtro mediante la función medianBlur() de OpenCV, con el fin de suavizarla antes de realizar la ecualización. Este paso permitió reducir significativamente el ruido sin afectar la nitidez de la imagen.

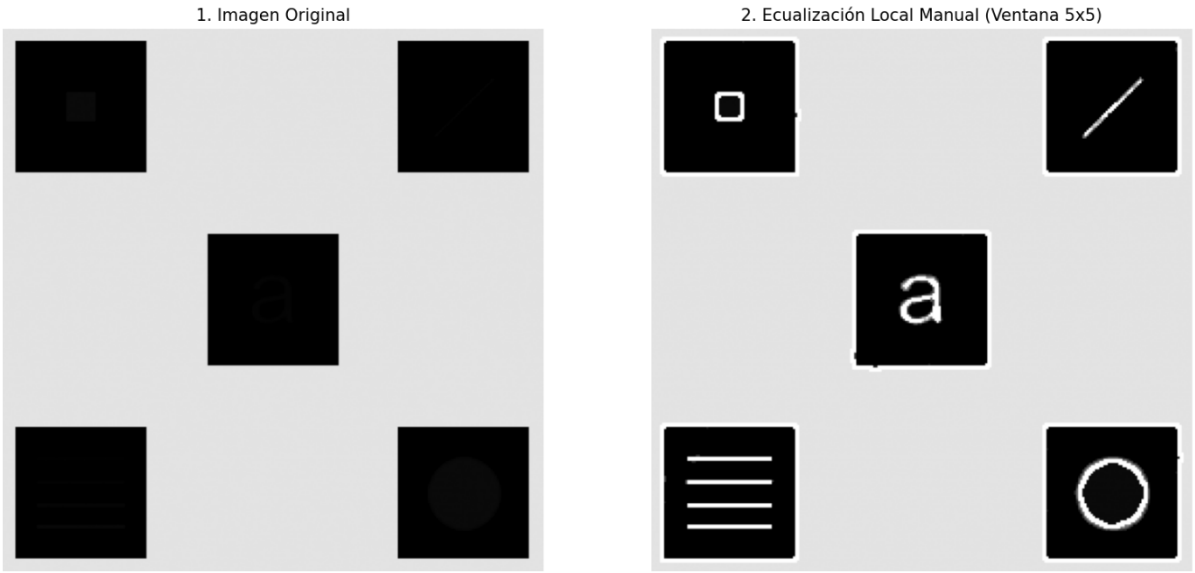


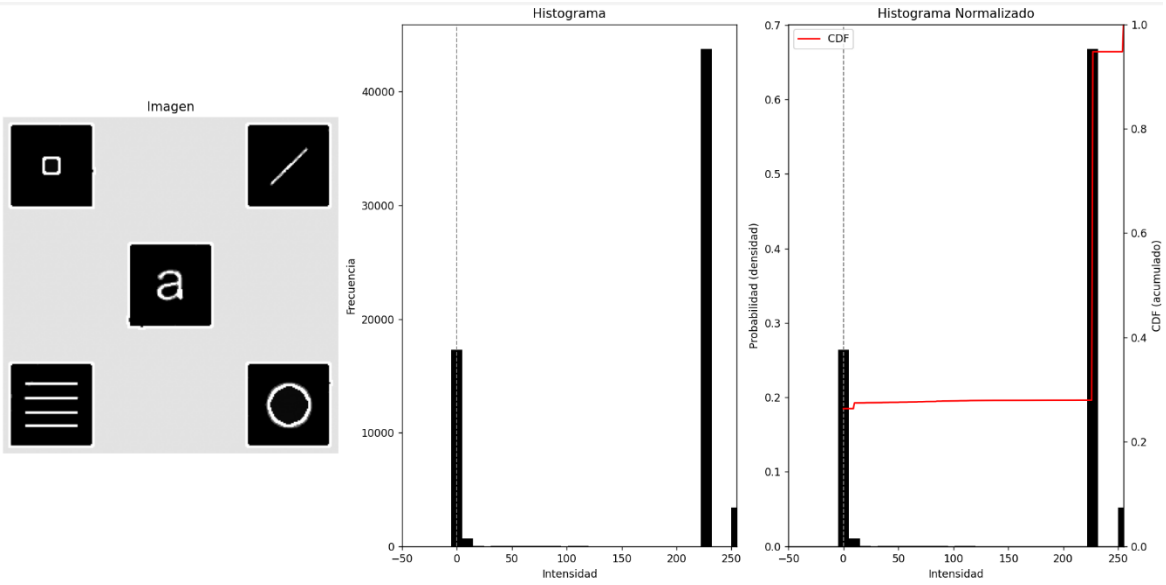
.
En la imagen de la derecha se aprecia una mejora en la definición y redondez de los bordes de los cuadros negros, producto de la función implementada.

Con la imagen ya suavizada, procedimos a evaluar distintos tamaños de ventana para analizar su influencia en el contraste de las figuras ocultas.

Ventana (5x5)

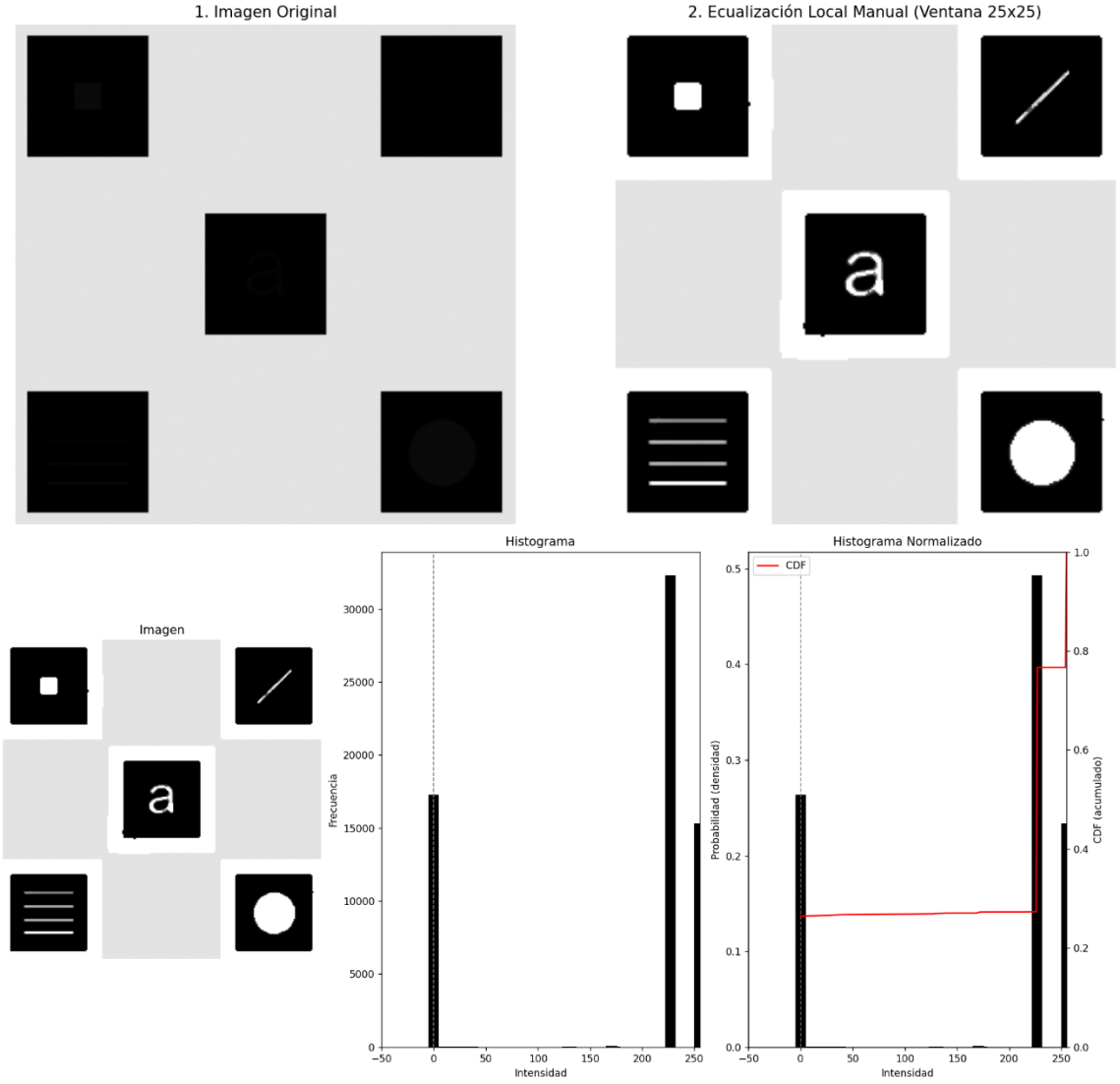
Una ventana pequeña produce un alto nivel de detalle, pero también genera mayor cantidad de ruido. En consecuencia, algunas de las figuras escondidas con mayor superficie no se muestran completamente uniformes.





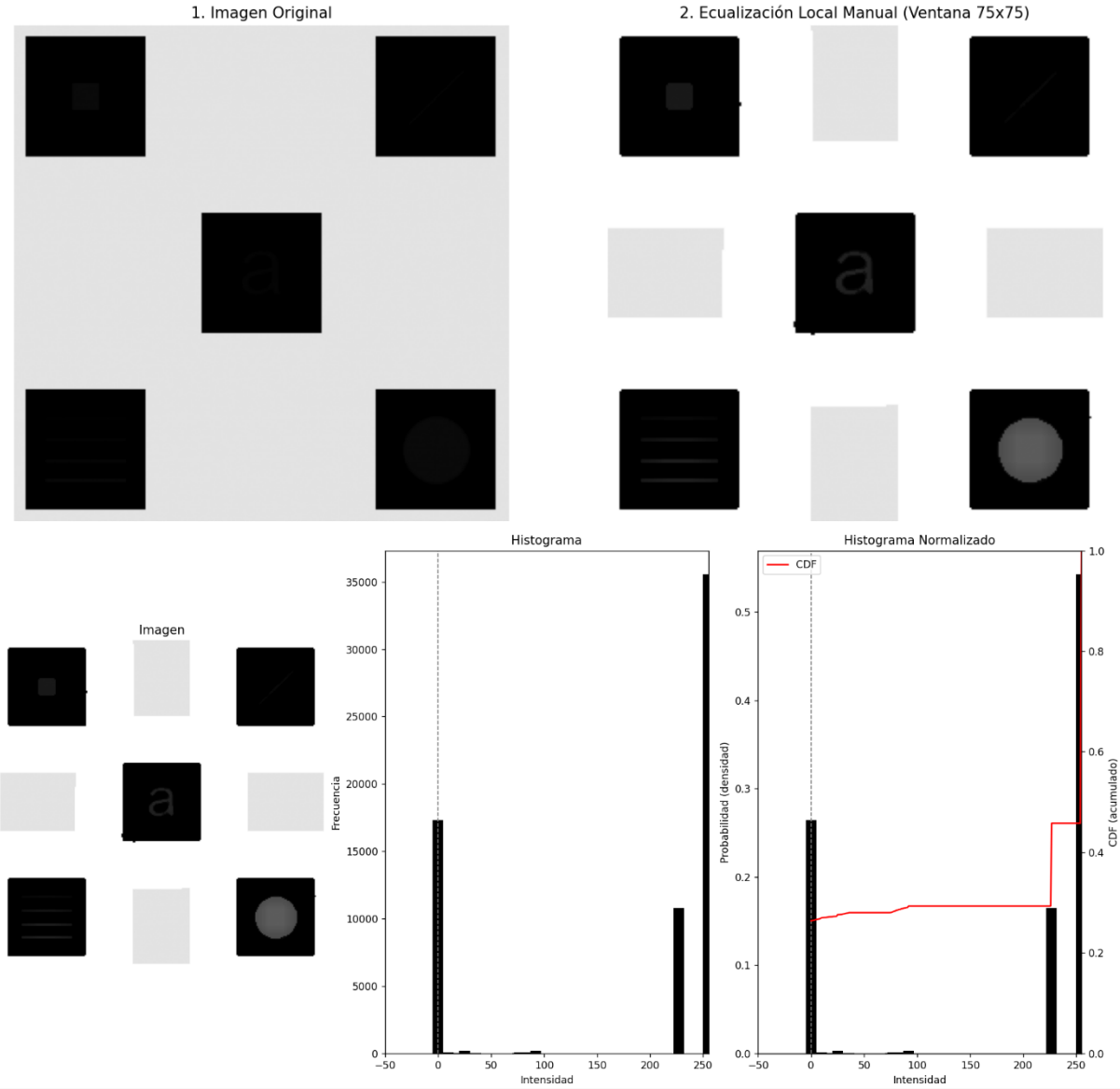
Ventana (25x25)

Una ventana de tamaño medio (en relación al tamaño total de la imagen) ofrece el mejor equilibrio entre detalle y ruido, logrando un contraste claro y nítido en los objetos dentro de los cuadros oscuros.



Ventana (75x75)

Finalmente, una ventana de gran tamaño genera un resultado comparable al de una ecualización global. Aunque el ruido disminuye, el contraste local se debilita y los objetos pierden nitidez.



En conclusión, la ecualización local combinada con un preprocesamiento de suavizado resultó ser la estrategia más efectiva para resaltar las zonas de interés, especialmente utilizando ventanas medianas (aproximadamente 25x25 píxeles), que brindan un buen equilibrio entre definición, contraste y nivel de ruido.

PROBLEMA 2 - VALIDACION DE FORMULARIO

Problemática a resolver

Se tiene como objetivo desarrollar un script en Python para validar automáticamente una serie de formularios en formato de imagen, basándose en un conjunto de restricciones específicas para cada campo.

Puntos a Desarrollar

El algoritmo debe abordar los siguientes puntos principales:

- A. Tomar la imagen del formulario como entrada y mostrar por pantalla el resultado de la validación para cada campo: OK (correcto) o MAL (incorrecto).
- B. Aplicar la validación de forma cíclica sobre un conjunto de cinco imágenes de formularios e informar los resultados.
- C. Crear una única imagen que informe visualmente qué personas completaron el formulario correctamente y cuáles incorrectamente. Debe incluir un "crop" del contenido del campo Nombre y Apellido de cada formulario, con un indicador de validación.
- D. Crear un archivo CSV para almacenar los resultados de cada validación. Cada fila debe corresponder a un formulario procesado, con columnas para un ID único y los ocho campos validados, registrando el resultado como OK o MAL en cada celda.

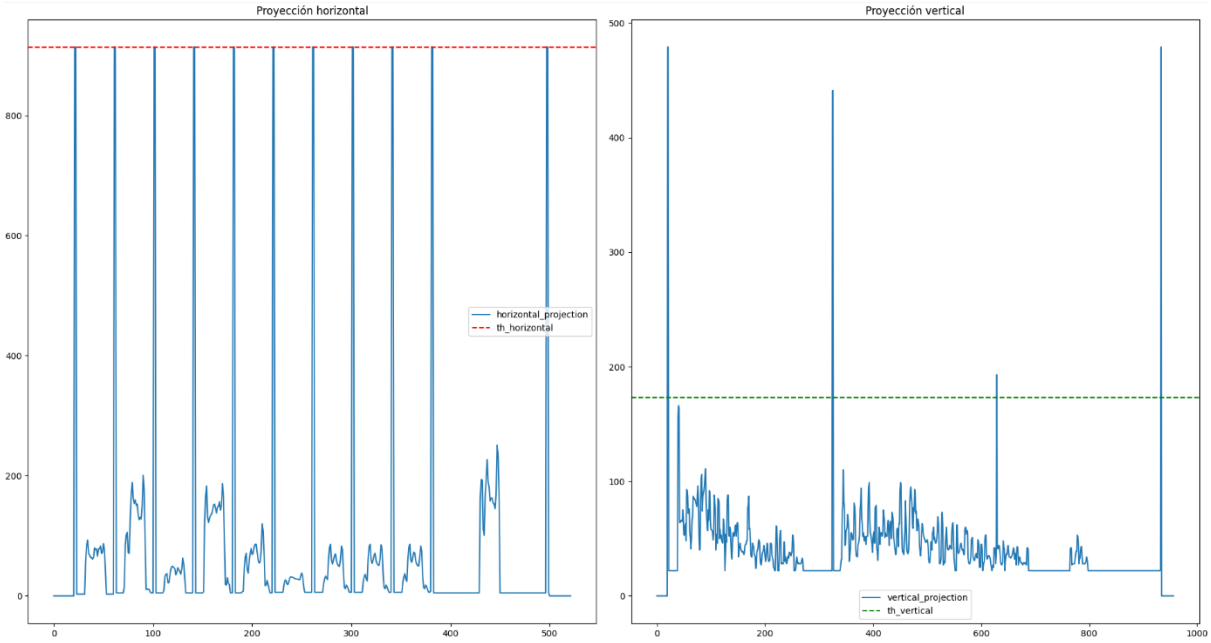
Resolución

Comenzamos analizando el formulario en blanco, cargando la imagen y aplicando un proceso de binarización. Luego, pasamos a la detección de líneas. El primer inconveniente que encontramos fue que todas las imágenes presentaban diferentes tonalidades de gris, lo que provocaba irregularidades en la binarización y generaba líneas que no eran completamente rectas.

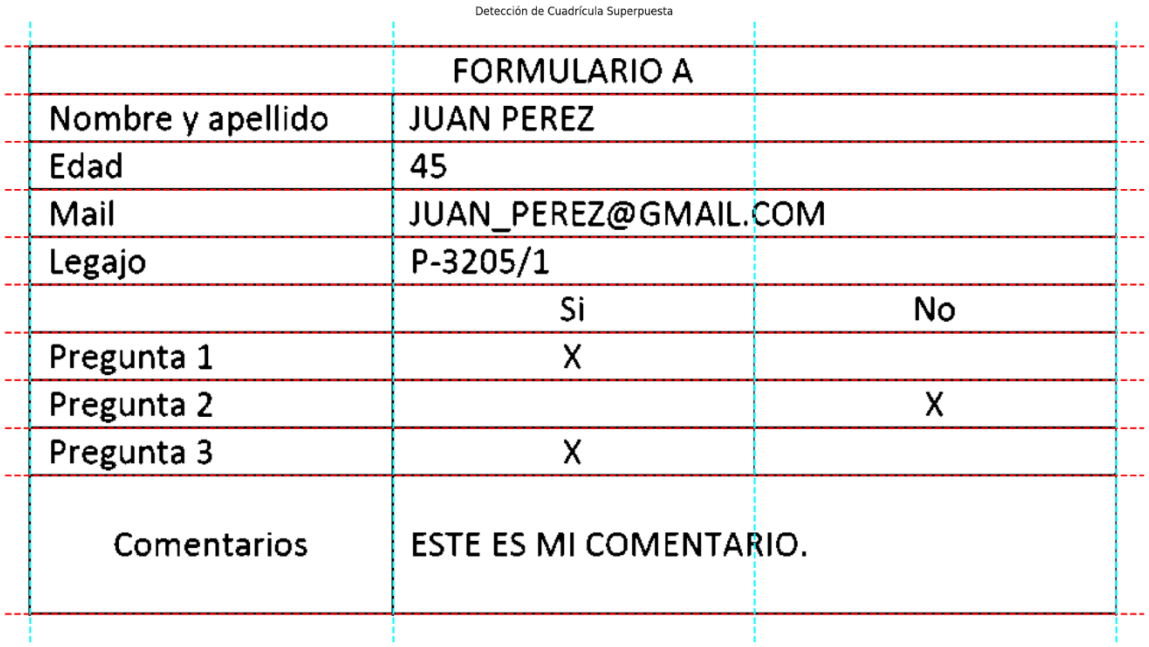
Para resolverlo, aplicamos el método de OTSU, que permitió establecer un umbral óptimo y uniforme, logrando una separación más precisa entre los píxeles blancos y negros. En el caso particular de las líneas verticales, algunas eran más largas que el recuadro del formulario y la última (de izquierda a derecha) resultaba más fina, esto no generó problemas significativos.

Para detectar las celdas realizamos una proyección horizontal y vertical de la imagen. Definimos un umbral en base a un percentil alto, ya que las líneas que queríamos identificar eran las que presentaban mayor cantidad de píxeles negros en la proyección.

El principal problema apareció en la proyección vertical, especialmente con la línea que separa las respuestas en la sección de preguntas. Dado que esta línea no era las más largas de la imagen, tuvimos que ajustar el percentil a un valor más bajo y visualizar los resultados mediante sliders, hasta encontrar un valor adecuado que evitara falsos positivos.

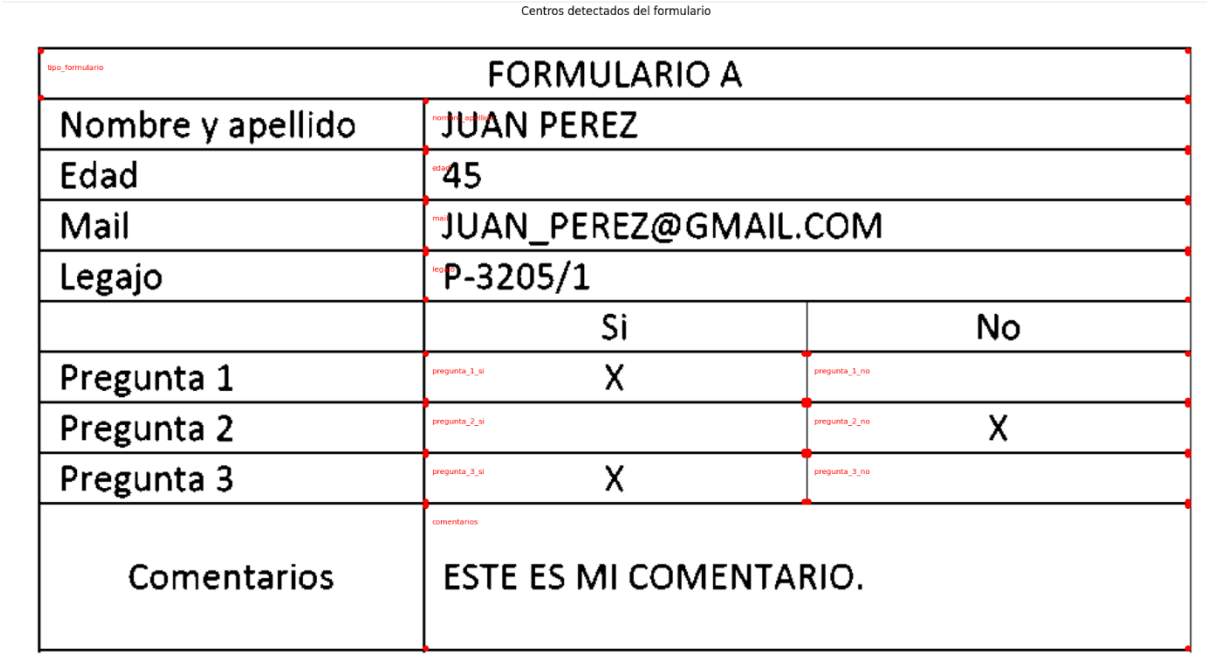


Para determinar las coordenadas de cada celda, analizamos los cambios entre píxeles negros y blancos utilizando las diferencias entre índices consecutivos con NumPy. Identificamos los “saltos” cuando la diferencia era mayor a uno, lo que correspondía a los espacios en blanco entre celdas.
Debido a que el grosor de las líneas podía ser irregular, tomamos como referencia la media entre índices consecutivos (para líneas horizontales o verticales).

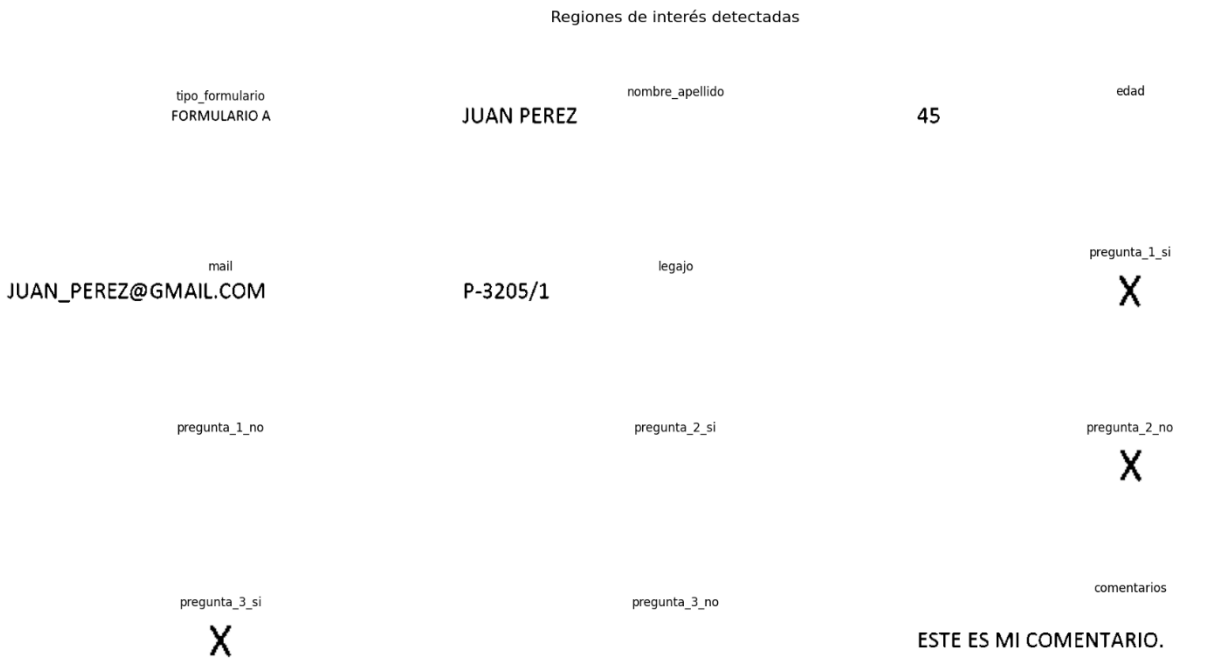


	FORMULARIO A	
Nombre y apellido	JUAN PEREZ	
Edad	45	
Mail	JUAN_PEREZ@GMAIL.COM	
Legajo	P-3205/1	
	Si	No
Pregunta 1	X	
Pregunta 2		X
Pregunta 3	X	
Comentarios	ESTE ES MI COMENTARIO.	

Posteriormente, calculamos las coordenadas exactas de cada celda y aplicamos un pequeño desplazamiento (1–2 píxeles) para evitar incluir las líneas negras dentro de la región de interés (ROI). Esto permitió obtener únicamente el contenido interior de cada celda.



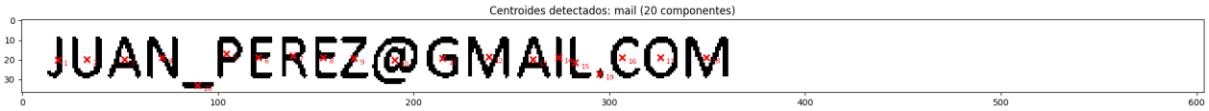
Con las coordenadas ya definidas, construimos un diccionario que almacena el array de la ROI correspondiente a cada celda del formulario.



A continuación, utilizamos la función connectedComponentsWithStats de OpenCV para identificar los componentes conectados dentro de cada ROI.
La detección de letras individuales funcionó correctamente, pero al intentar detectar

palabras completas surgió un problema: no existía una función directa para agrupar letras en palabras.

Para resolverlo, empleamos los centroides de los objetos detectados. Calculamos las distancias entre centroides consecutivos y usamos una mediana de referencia para identificar posibles espacios entre palabras.
Sin embargo, asumimos erróneamente que las distancias entre letras con espacios siempre serían mayores que entre letras consecutivas, lo cual no se cumplía en todos los casos.



La solución fue definir un umbral constante por celda para distinguir palabras, determinado de forma empírica mediante pruebas manuales hasta encontrar valores adecuados.

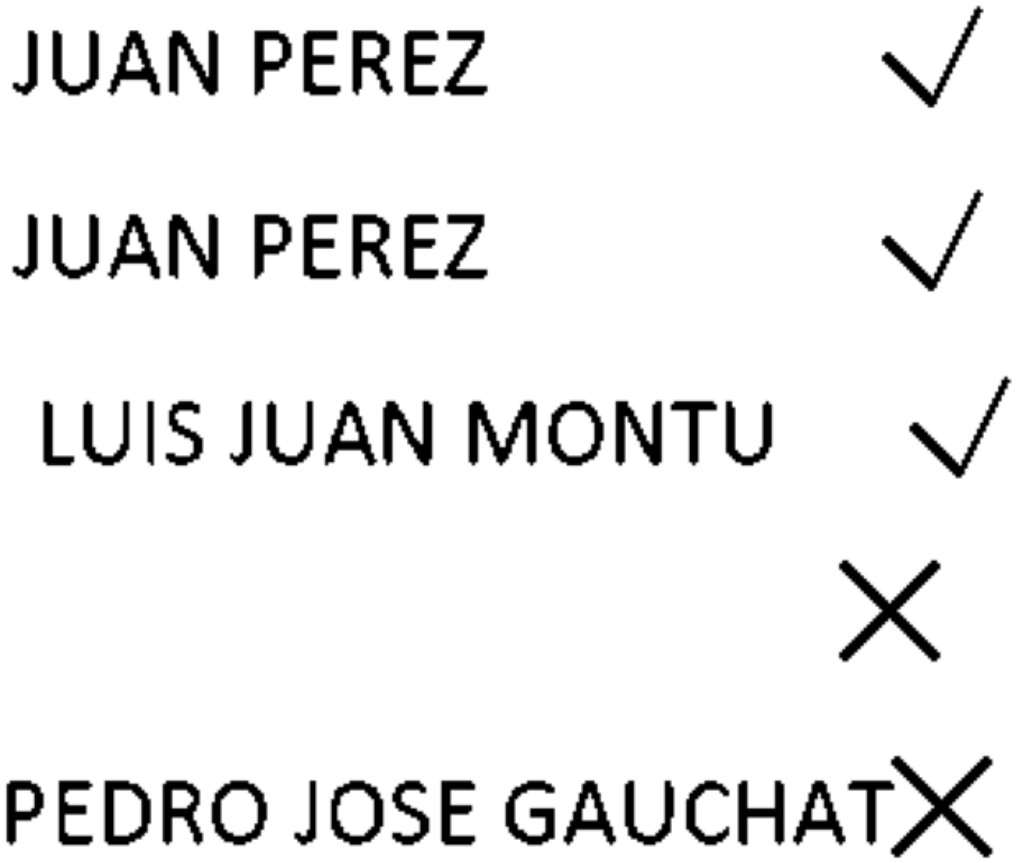
Una vez que logramos contar letras y palabras correctamente, desarrollamos una función genérica de validación que aplicamos a cada formulario.
Los resultados se almacenaron en un diccionario, que luego imprimimos por pantalla cumpliendo así con el punto A del enunciado.

```
Formulario formulario_01.png:
>>> for campo, resultados in resultado_validaciones.items():
...     resultado = "OK" if resultados["es_valido"] else "MAL"
...     print(f"{campo}: {resultado}")
... # Punto C
...
nombre_apellido: OK
edad: OK
mail: OK
legajo: OK
pregunta_1: OK
pregunta_2: OK
pregunta_3: OK
comentarios: OK
```

Para el siguiente paso, realizamos los crops correspondientes al campo “Nombre y Apellido”.
El primer inconveniente fue que las imágenes tenían dimensiones distintas, lo que impedía unirlas correctamente en un mismo canvas. Lo solucionamos tomando la imagen de mayor tamaño como base para generar un canvas en blanco, sobre el cual insertamos los recortes de cada formulario.

El segundo problema ocurrió en el formulario 4, donde el campo de nombre estaba vacío. En ese caso, la función connectedComponentsWithStats devolvía únicamente el centroide del fondo y otro con valor NaN, provocando un error al intentar ubicar la tilde o la cruz.

Resolvimos este inconveniente utilizando el centroide del fondo como referencia para posicionar el indicador correspondiente.



Finalmente, con los diccionarios de resultados ya estructurados, creamos el archivo CSV solicitado. Generamos una lista con los encabezados y otra con las filas de resultados, y luego empleamos la función de escritura de CSV en Python para guardar los datos.
Este paso se ejecutó correctamente, sin presentar inconvenientes.