“Procesamiento de imágenes - Trabajo práctico 1”

Integrantes:

-Juan Morales
-Genaro Canciani
-Malena Ruppen

Profesores: Gonzalo Sad, Juan Manuel Calle, Joaquín Allione.

21/10/2025

Ejercicio 1
Ecualización local

Objetivo: El objetivo principal de este ejercicio es implementar y analizar la técnica de **Ecualización Local de Histograma** (LHE, por sus siglas en inglés). A diferencia de la ecualización global, que aplica una única transformación a toda la imagen basándose en su histograma general, la ecualización local adapta la transformación de contraste a regiones específicas.
La implementación se realiza mediante un método de **ventana deslizante**. Para cada píxel de la imagen original, se define una vecindad (o ventana) de tamaño M*N centrado en dicho píxel, siendo M el alto de la ventana definida(cantidad de filas seleccionadas) y N el ancho de la ventana(cantidad de columnas seleccionadas). Luego, se calcula y aplica la ecualización de histograma exclusivamente a los píxeles dentro de esta ventana. El valor del píxel central en la ventana ecualizada resultante se asigna al píxel correspondiente en la imagen de salida.
Este ejercicio utiliza el script *"ej1.py"* para aplicar esta técnica sobre la imagen *"Imagen_con_detalles_escondidos.tif",* explorando cómo varía el resultado al modificar el tamaño de la ventana.

Resolución: El script mencionado implementa una solución utilizando librerías y estrategias observadas en clase, que se mencionan a continuación.

Técnicas y herramientas utilizadas:
Librerías:

- OpenCV para lectura de imagen, padding y ecualización local.
- NumPy para manipulación matricial.
- Matplotlib para visualización de resultados.

Transformación utilizada**:** ecualización de histograma (equalizeHist) sobre ventanas locales.

La implementación detallada de esta transformación comienza con la definición de la función "ecualizacion_local", que toma como argumentos la imagen en escala de grises y las dimensiones de la ventana (M y N). El primer paso crucial dentro de la función es el manejo de los bordes: se calcula el *padding* necesario (M//2 y N//2) y se utiliza la función cv2.copyMakeBorder con el método BORDER_REPLICATE para crear una versión de la imagen original que tiene un padding proporcional al tamaño de la ventana, lo que garantiza que la ventana de procesamiento pueda centrarse incluso en los píxeles del borde sin salirse de los límites de la imagen. Se observa a continuación el código utilizado.

```
# Padding para los bordes
pad_M, pad_N = M // 2, N // 2
img_padded = cv2.copyMakeBorder(img,pad_M, pad_M, pad_N, pad_N, borderType=cv2.BORDER_REPLICATE)
```

.

Luego, se inicializa una matriz de salida (salida) del mismo tamaño que la imagen original, utilizando np.zeros_like, que se utilizará luego para ir mapeando los píxeles obtenidos en cada ecualización a la misma y luego retornarla al usuario.

El núcleo del proceso consiste en un doble bucle for que itera sobre cada coordenada (i, j) de la imagen *original*. En cada iteración, se extrae una sub-región o "ventana" de tamaño M x N de la imagen con padding, asegurando que la ventana esté centrada en el píxel (i, j) actual. Para eso se suma los valores de M y N definidos previamente a los índices para garantizar que la ventana seleccionada tenga las proporciones elegidas por el usuario, como se puede ver en la siguiente imagen:

```
# Extraer ventana
ventana = img_padded[i:i+M, j:j+N]
```

.

Sobre esta ventana extraída, se aplica la transformación cv2.equalizeHist, la cual calcula y aplica una ecualización de histograma completa pero limitada *únicamente* a los píxeles de esa pequeña región. Esta técnica busca redistribuir los niveles de intensidad para que la imagen resultante tenga un histograma lo más uniforme posible, cubriendo mejor todo el rango dinámico [0, 255] y aumentando el contraste de la imagen. De la ventana_ecualizada resultante, el script no conserva toda la región, sino que extrae exclusivamente el valor del píxel central, ubicado en [pad_M, pad_N], obtenido al realizar la operación M // 2 y N // 2, y lo asigna a la imagen de salida en la posición (i, j). Este proceso de "ventana deslizante" se repite para cada píxel de la imagen.
Una vez terminado el proceso se retorna el contenido de la variable *"salida"*, que contiene la imagen con la ecualización realizada.

Problemas detectados:

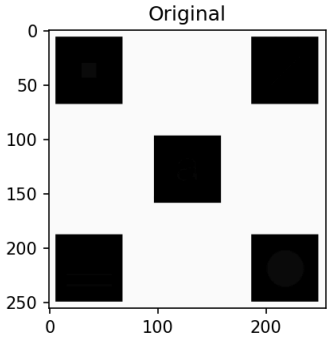
1. Manejo de bordes: Al aplicar ventanas en los bordes, algunas coordenadas quedaban fuera de rango.
2. Histograma sobre la imágen completa: Durante la primera implementación, aunque se definía correctamente un vecindario alrededor de cada píxel, la ecualización no se realizaba de manera local. Esto se debía a que el histograma se calculaba sobre toda la imagen (img) en cada iteración, en lugar de sobre la ventana correspondiente a ese píxel. Como resultado, todos los píxeles se transformaban utilizando la misma función de distribución acumulada (CDF), lo que equivalía en la práctica a realizar una **ecualización global repetida**.
   Como consecuencia el resultado no variaba con el tamaño de la ventana, no se apreciaba realce localizado de contraste y se realizaba un cálculo costoso sin obtener ningún beneficio adicional.
   Para solucionar este problema se modificó el cálculo del histograma para que utilizara únicamente la ventana local. Esto permitió que cada píxel se transformara según la distribución de intensidades de su entorno, logrando finalmente la **ecualización local de histograma** deseada.

Resultados obtenidos:

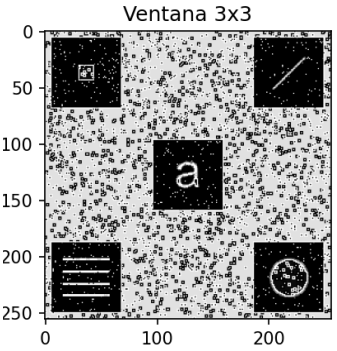
A continuación, se muestran los resultados de aplicar ecualización local de histograma sobre la imagen original utilizando diferentes tamaños de ventana.

La imagen original presenta **zonas oscuras con poco contraste**, donde muchos detalles permanecen ocultos o apenas perceptibles. Al aplicar la técnica de ecualización local, se logra realzar progresivamente distintas regiones de la imagen en función del tamaño del vecindario utilizado:

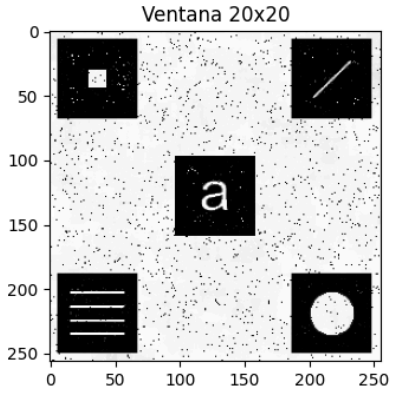
Imagen original:
Presenta bajo contraste global, especialmente en las regiones oscuras. Las estructuras internas (líneas, texto, contornos) están presentes pero no son visibles con claridad.



Ventana 3×3:
Se observa un **fuerte realce local** de los detalles. Las zonas oscuras revelan estructuras internas que antes no eran visibles. Sin embargo, al trabajar con un vecindario tan pequeño, también se **amplifica el ruido**, generando un patrón granular en áreas homogéneas de la imagen.

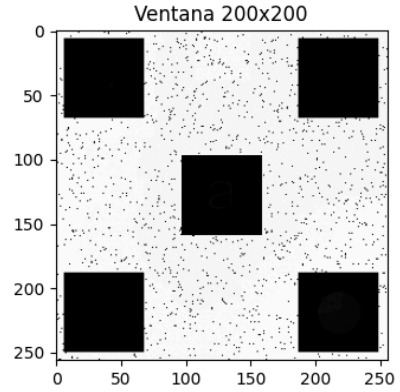


Ventana 20×20:
Se obtiene un **buen equilibrio entre detalle y contraste**. El ruido aleatorio se reduce considerablemente respecto a la ventana 3×3, pero aún se preservan detalles finos en las zonas oscuras. Es un tamaño de ventana adecuado para mejorar la visibilidad sin deteriorar demasiado la calidad visual.



Ventana 200×200:

Al utilizar un vecindario tan grande, que se acerca al tamaño de la imagen procesada, el comportamiento de la ecualización local se aproxima al de una ecualización global. El contraste de la imagen se mantiene similar al observado en la imagen original, aunque permitiendo apreciar un poco más los detalles de las partes oscuras, aunque la diferencia sea sutil.



Se puede concluir que al aumentar el tamaño de la ventana, se reduce la sensibilidad local y se incrementa el efecto global. Ventanas pequeñas realzan detalles finos (a costa de más ruido), mientras que ventanas grandes mejoran el contraste general (pero suavizan detalles).

Conclusión:

Este ejercicio permite comprender de manera práctica la diferencia entre ecualización global y local, y cómo el **tamaño de ventana** es un parámetro crítico para controlar el grado de realce local en una imagen. Este análisis es especialmente útil para aplicaciones donde se requiere mejorar la visibilidad de detalles sin introducir ruido excesivo, como en imágenes médicas, inspección industrial o visión artificial.

Ejercicio 2
Validación de formularios

Objetivo: En El Ejercicio 2 se plantea el desafío de crear un script en Python capaz de procesar automáticamente un lote de formularios que han sido completados y guardados como imágenes (archivos formulario_<id>.png). El objetivo principal es validar el contenido de cada formulario, verificando que la información ingresada en cada campo (Nombre y Apellido, Edad, Mail, Legajo, Preguntas 1, 2 y 3, y Comentarios) se adhiera a un conjunto específico de reglas y restricciones. Estas reglas definen, por ejemplo, el número de palabras, la longitud de caracteres, y las condiciones de marcado para las casillas de "Si" y "No". El ejercicio requiere que el script no solo procese cíclicamente todas las imágenes e informe por terminal el estado ("OK" o "MAL") de cada campo, sino que también identifique el tipo de formulario (A, B o C) y genere dos salidas consolidadas: una única imagen que muestre los nombres de los aplicantes con un indicador visual de si su formulario fue correcto o incorrecto , y un archivo CSV que registre los resultados detallados de la validación para cada formulario.

Resolución: De igual manera que con el anterior, el script mencionado implementa una solución utilizando librerías y estrategias observadas en clase, que se mencionan a continuación.

Técnicas y herramientas utilizadas:

Librerías:

- OpenCV para aplicar umbralado y detectar componentes conectados.
- NumPy para manipulación matricial.
- Matplotlib para visualización de resultados.
- Csv para generar el archivo de salida con los resultados obtenidos.

La implementación detallada de esta validación comienza con un bucle principal que itera sobre la lista de archivos de formularios. El primer paso crucial dentro del bucle es la **detección dinámica de la grilla** del formulario. Para esto, se carga la imagen en escala de grises y se binariza usando un umbral de 120. Se eligió este umbral ya que no había en la imagen una gran variedad de valores de grises, puesto que la misma tiene solo el negro de los caracteres y las celdas y el blanco del fondo.

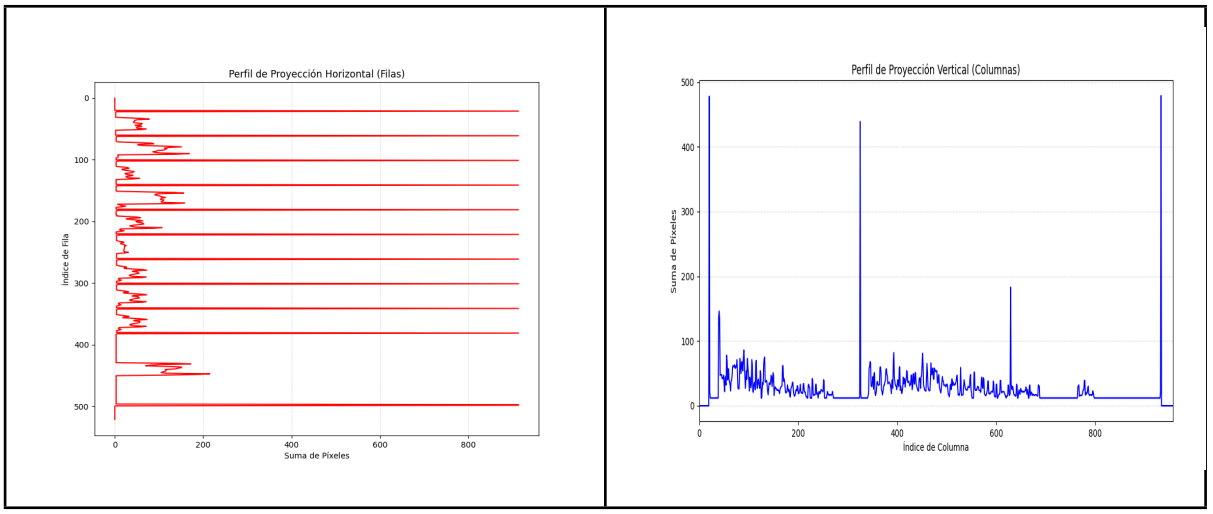
Imagen original:

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.	

Imagen binarizada:

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.	

A continuación, se aplica una proyección de histograma sumando los valores de los píxeles a lo largo de cada fila (.sum(axis=1)) y cada columna (.sum(axis=0)). Al umbralizar estos vectores de suma, en la variable "rows_detect" y "cols_detect" obtiene un array con valores de verdadero o falso para aquellas filas y columnas que superan el valor del umbral. Se utiliza sobre esos vectores la estrategia de usar las funciones where y diff de numpy para obtener los índices de las filas y columnas en los que los que hay un cambio de valores. Se visualizan los vectores de las sumatorias en las siguientes imágenes:



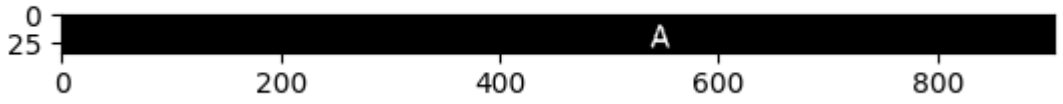
Luego, basándose en las coordenadas de las líneas detectadas (horizontal_lines y vertical_lines), el script procede a **extraer las Regiones de Interés (ROIs)**. Esto se logra mediante el slicing de la matriz de la imagen original, recortando la porción exacta que corresponde a cada celda de entrada de datos (ej. celda_nombre, celda_edad, celda_p1s, etc).

El núcleo del análisis reside en la función analizar_celda, que recibe una de estas ROIs o celdas. Dentro de esta función, la celda se binariza usando el método de Otsu (cv2.THRESH_OTSU) y se aplica la función cv2.connectedComponentsWithStats. Esta técnica es fundamental ya que identifica cada conjunto de píxeles conexos (es decir, cada carácter) como un componente individual y devuelve sus estadísticas, como su área y posición.

Sobre estas estadísticas, el script realiza un filtrado en donde descarta componentes cuya área sea muy pequeña (considerados ruido) o muy grande (probablemente restos de las líneas de la grilla) usando th_min_area y th_max_area. La cantidad de componentes restantes se considera el número de caracteres válidos. Adicionalmente, para estimar la cantidad de palabras, ordena los caracteres por su posición horizontal y cuenta un "espacio" (incrementando palabras) cada vez que la distancia entre el final de un carácter y el inicio del siguiente supera un umbral (space_threshold). La función retorna finalmente la cantidad de caracteres y palabras detectadas.

Los conteos de todas las celdas se envían a la función validar_formulario. Esta función no es más que una serie de reglas implementadas con lógica booleana. Compara los números obtenidos con las restricciones del ejercicio (ej. (words_nombre >= 2) and (chars_nombre <= 25)). Para las preguntas, utiliza el operador XOR (^) para garantizar que *solo una* de las dos casillas (Si o No) tenga exactamente un carácter.

Paralelamente a la validación de campos, el script identifica el tipo de formulario (A, B o C) mediante la función detectar_tipo_formulario, la cual recibe la Región de Interés (ROI) de la celda de encabezado. La estrategia se basa en la topología de las letras, específicamente en el conteo de sus "huecos" internos. Para ello, la celda se binariza con el método de Otsu (invirtiendo el resultado para obtener letras blancas sobre fondo negro) y se utiliza cv2.connectedComponentsWithStats para localizar todos los caracteres. Tras un filtrado de componentes basado en área para descartar ruido o bordes, el script aísla el componente situado más a la derecha (asumiendo que es la letra A, B o C). Finalmente, sobre una máscara de este único componente, se aplica cv2.findContours con el modo jerárquico cv2.RETR_CCOMP para contar cuántos contornos internos (huecos) posee. El formulario se clasifica como "C" si detecta cero huecos, "A" si detecta uno, y "B" si detecta dos. Se ve en la imagen un ejemplo de máscara detectada por la función en el formulario 1.



Una vez terminado el proceso de validación para los formularios, los resultados (un diccionario de valores True/False) se almacenan. Finalmente, fuera del bucle principal, el script genera las tres salidas requeridas: imprime los resultados por consola, utiliza matplotlib para crear la imagen consolidada (coloreando los títulos de verde o rojo según el estado) y emplea el módulo csv para escribir el archivo resultados_formularios.csv, traduciendo los valores True/False a "OK" y "MAL" como se ve en la imagen.

resultados_formularios.csv > data

```
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,MAL,MAL,OK,MAL,MAL,MAL
05,OK,MAL,OK,OK,MAL,MAL,MAL,OK
```

Problemas detectados:

1. Detección de líneas de segmentación: Durante el procesamiento del formulario, no se detectó correctamente la columna correspondiente a las opciones “Sí”, “No”. La columna era más angosta que las otras y el umbral de detección de columnas estaba ajustado suponiendo que todas las columnas tenían un ancho similar.
   Como consecuencia esa columna no generaba un pico suficientemente grande para superar el umbral.
2. Ajuste de área de componentes conectadas: Se implementó un filtrado por área, descartando las CC con un área muy pequeña (th_min_area) y aquellas con un área excesivamente grande (th_max_area), que probablemente serían restos de bordes de la imagen o texto mal segmentado.
3. Estimación de palabras (ajuste del espacio entre caracteres): La estimación de la cantidad de palabras se realiza midiendo la distancia horizontal entre los *bounding boxes* de caracteres consecutivos. Se asumió que todos los caracteres (letras, números, símbolos) tienen una apariencia visual y un tamaño de fuente uniformes, lo que permitió establecer un parámetro fijo de separación de palabras (space_threshold = 6 píxeles). Este umbral se calibró para ser mayor que el espaciado típico entre letras (interletraje) dentro de una palabra, y menor que el espaciado mínimo entre palabras.

Conclusión:

Este ejercicio permite comprender cómo aplicar técnicas de procesamiento de imágenes para la **extracción y validación automática de información estructurada** en formularios. A través del análisis de componentes conectados y la detección de celdas mediante umbralado, se logra automatizar la evaluación de campos y reglas predefinidas. Esta práctica resulta especialmente útil en aplicaciones de **digitalización de documentos, control de calidad y reconocimiento óptico de caracteres**, donde se busca reducir errores humanos y optimizar el procesamiento masivo de datos visuales.