UNIVERSIDAD NACIONAL DE ROSARIO
FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Trabajo Práctico N° 1

PROCESAMIENTO DIGITAL DE IMÁGENES I

INFORME

Ecualización Local de Histograma
Validación de Formulario

Carrera: Tecnicatura Universitaria en Inteligencia Artificial

Alumnos: *Demarré, Lucas*

Donnarumma, Cesar Julian

Menescaldi, Brisa

Mussi, Miguel

Ciclo: *2023*

TABLA DE CONTENIDOS

PARTE I: INTRODUCCIÓN

Problema 1 – Ecualización local de histograma

La técnica de ecualización del histograma se puede extender para un análisis local, es decir, se puede realizar una ecualización local del histograma. El procedimiento sería definir una ventana cuadrada o rectangular (vecindario) y mover el centro de la ventana de pixel a pixel. En cada ubicación, se calcula el histograma de los puntos dentro de la ventana y se obtiene de esta manera, una transformación local de ecualización del histograma. Esta transformación se utiliza finalmente para mapear el nivel de intensidad del pixel centrado en la ventana bajo análisis, obteniendo así el valor del pixel correspondiente a la imagen procesada. Luego, se desplaza la ventana un pixel hacia el costado y se repite el procedimiento hasta recorrer toda la imagen.

Esta técnica resulta útil cuando existen diferentes zonas de una imagen que poseen detalles, los cuales se quiere resaltar, y los mismos poseen valores de intensidad muy parecidos al valor del fondo local de la misma. En estos casos, una ecualización global del histograma no daría buenos resultados, ya que se pierde la localidad del análisis al calcular el histograma utilizando todos los pixeles de la imagen.

Desarrolle una función para implementar la ecualización local del histograma, que reciba como parámetros de entrada la imagen a procesar, y el tamaño de la ventana de procesamiento (M x N). Utilice dicha función para analizar la imagen que se muestra en Fig. 1 e informe cuales son los detalles escondidos en las diferentes zonas de la misma. Analice la influencia del tamaño de la ventana en los resultados obtenidos.

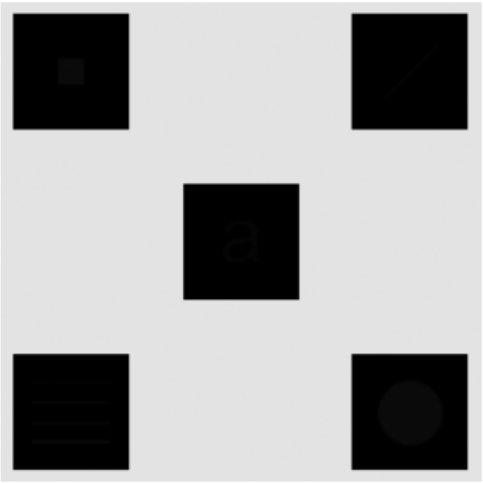


Figura 1 - Imagen con detalles en diferentes zonas.

AYUDA:

Con la siguiente función, puede agregar una cantidad fija de pixels a una imagen: ***cv2.copyMakeBorder(img, top, bottom, left, right, borderType)***, donde top, bottom, left y right son valores enteros que definen la cantidad de pixels a agregar arriba, abajo, a la izquierda y a la derecha, respectivamente, y borderType define el valor a utilizar. Por ejemplo, ***borderType=cv2.BORDER_REPLICATE*** replica el valor de los bordes.

Problema 2 – Validación de formulario

En la Figura 2 se muestra el esquema de un formulario (imagen formulario_vacio.png), con sus respectivos campos. La primera fila define el tipo de formulario (A, B o C), luego hay 4 campos para completar datos personales, luego 3 preguntas que deben responderse por SI o por NO, y por último un campo de comentarios libres.

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

Figura 2 – Esquema del formulario.

Se tiene una serie de formularios completos, en formato de imagen, y se pretende validar cada uno de ellos, corroborando que cada uno de sus campos cumpla con las siguientes restricciones:

1. Nombre y apellido: Debe contener al menos 2 palabras y no más de 25 caracteres en total.
2. Edad: Debe contener 2 o 3 caracteres.
3. Mail: Debe contener 1 palabra y no más de 25 caracteres.
4. Legajo: 8 caracteres formando 1 sola palabra.
5. Preguntas: se debe marcar con 1 caracter una de las dos celdas SI y NO. No pueden estar ambas vacías ni ambas completas.
6. Comentarios: No debe contener más de 25 caracteres.

Asuma que todos los campos ocupan un solo renglón (el de comentarios también), y que se utilizan solo las siguientes letras mayúsculas, números y símbolos:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 @ . - /_

En la Figura 3a se muestra un ejemplo donde los campos del formulario están todos correctamente cargados, mientras que en la Figura 3b se muestra otro ejemplo donde todos los campos están mal cargados.

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)

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.	

(b)

Figura 3 – Ejemplos de carga de formularios. (a) Todos los campos bien cargados. (b) Todos los campos mal cargados.

Desarrolle un algoritmo para validar los campos del formulario. Debe tomar como entrada la imagen del mismo y mostrar por pantalla el estado de cada uno de sus campos. Por ejemplo:

Nombre y apellido: OK
Edad: OK
Mail: MAL
Legajo: MAL
Pregunta 1: OK
Pregunta 2: MAL
Pregunta 3: OK
Comentarios: OK

Utilice el algoritmo desarrollado para evaluar las imágenes de formularios completos (archivos formulario_xx.png) e informe los resultados obtenidos.

AYUDAS:

1) Existen varias formas de detectar las celdas donde se cargan los datos, una de ellas es detectando las coordenadas de las líneas verticales y horizontales que dividen el formulario en filas y columnas. Para ello, una opción es primero umbralar la imagen img_th = img<th y luego sumar el valor de los pixels que están en cada columna para detectar las columnas img_cols = np.sum(img_th_ones,0) y sumar el valor de los pixels que están en cada fila para detectar las filas img_rows = np.sum(img_th_ones,1). Luego, dado que en dichas líneas existen muchos más pixels que en las demás partes del formulario, se puede definir un umbral acorde (uno para las filas y otro para las columnas) y detectar así las posiciones de las mismas. Por ejemplo: img_rows_th =img_rows>th_row. Tenga en cuenta que las líneas pueden tener más de un pixel de ancho, por lo cual, quizás deba encontrar el principio y el fin de las mismas en la variable img_rows_th.

2) Una vez obtenida las celdas, una posible forma de obtener los caracteres dentro de la misma, es obteniendo las componentes conectadas dentro de la misma:

cv2.connectedComponentsWithStats(celda_img, 8, cv2.CV_32S).

Tenga especial cuidado que no hayan quedado pixeles de las líneas divisorias de la tabla dentro de la celda. Una posible forma de evitar este problema, es eliminar las componentes conectadas de área muy chica, definiendo un umbral: ix_area = stats[:,-1]>th_area y luego stats = stats[ix_area,:].

PROBLEMA I

Funcionamiento del código

El código es una implementación de la ecualización local de histograma utilizando la biblioteca OpenCV (cv2) y Matplotlib en Python. La ecualización local de histograma es una técnica que mejora el contraste de una imagen al ajustar el histograma de intensidad en pequeñas regiones locales de la imagen en lugar de toda la imagen.

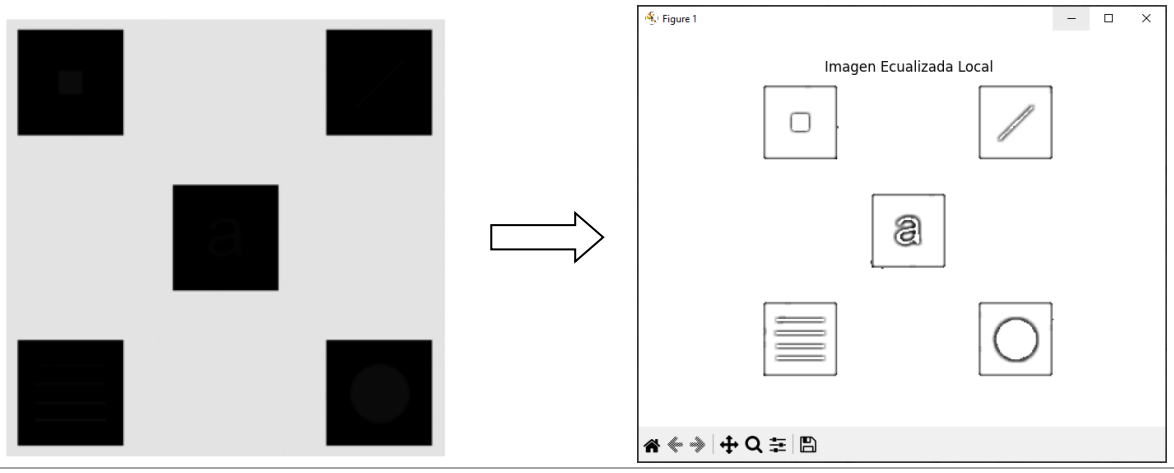
Detalle paso a paso:

- Importa las bibliotecas necesarias: OpenCV (`cv2`) para el procesamiento de imágenes y Matplotlib (`plt`) para mostrar la imagen resultante.
- Define una función llamada `local_histogram_equalization` que toma una imagen, así como el tamaño de la ventana (ventana deslizante) utilizada para realizar la ecualización local del histograma.
- Calcula la mitad de la altura y la anchura de la ventana para asegurarse de que la ventana esté centrada en cada píxel.
- Aplica un filtro de mediana con un tamaño de kernel de 3x3 a la imagen original. Esto se hace para reducir el ruido en la imagen antes de realizar la ecualización del histograma local.
- Agrega bordes a la imagen para evitar problemas cuando la ventana se desplace cerca de los bordes de la imagen. La opción `cv2.BORDER_REPLICATE` significa que los bordes se replicarán para completar la imagen.
- Luego, se realiza un bucle sobre cada píxel de la imagen original. Para cada píxel, se extrae una ventana local alrededor de ese píxel utilizando las dimensiones de la ventana especificadas.
- Se calcula el histograma de la ventana local utilizando `cv2.calcHist`.
- Se calcula la función acumulativa del histograma (CDF) normalizada.
- Se calcula el valor transformado para el píxel central de la ventana local utilizando el CDF normalizado.
- Finalmente, se asigna este valor transformado al píxel correspondiente en la imagen original.
- La función devuelve la imagen resultante después de aplicar la ecualización local del histograma.

- Se carga una imagen en escala de grises ('Imagenes\Problema_1\Imagen_con_detalles_escondidos.tif').
- Se especifica el tamaño de la ventana (5x5 en este caso) y se llama a la función para aplicar la ecualización local del histograma.
- Se muestra la imagen resultante utilizando Matplotlib con un título y los ejes desactivados.

Ejemplo de ejecución y salida

Luego de la ejecución del algoritmo sobre la imagen “Imagen_con_detalles_escondidos.tif” se mostrará por pantalla el siguiente resultado:



Problemas y resolución

Para plantear la solución hay que entender el problema, el principal obstáculo fue dado por el entendimiento del concepto de que es una ecualización local más que por la complejidad del código. A partir de entender el concepto, sobre que es primero una ecualización local, que esta se hace en todos los pixeles de la imagen uno por uno y no todos a la vez como en una ecualización global, te permite sacar la primera parte del código que es el de crear dos bucles que recorren todos los pixeles de la imagen y aplicar justamente en estos la ecualización.

Pero para hacer una ecualización es necesario tener otros pixeles que le brinden información al histograma, si solo usamos un pixel y se hace la ecualización, todos los pixeles se saturan y no conseguiríamos resaltar los detalles en la imagen. Para esto había que crear una ventana de MxN tamaño para que se utilicen esos pixeles como información tanto en el histograma como la ecualización que se hace a posterior para definir la intensidad del pixel central que se está recorriendo. Con eso ya se sabe que después de cada ecualización hay que definir el pixel dentro del código, quedando entonces, hasta ahora, una función que en cada iteración de los dos primeros bucles que recorren la imagen, es decir, en cada pixel recorrido, se crea una ventana de un tamaño dado, se hace un histograma y luego su ecualización para que, después de normalizar dicha ecualización, se defina el pixel que se está recorriendo y así con todos los pixeles de la imagen.

Después lo demás es prácticamente algo que ya te dice el ejercicio, se crea una función que reciba como parámetros la imagen y el tamaño de la ventana, descomprimís la imagen para tener sus dimensiones para los bucles siguientes y después de hacer la ecualización en cada pixel, se devuelve la imagen.

Al hacer esto aparece el primer problema, el código no funciona ya que determinados pixeles tienen un tamaño de ventana menor al que deberían tener, haciendo que el código no funcione. Al investigar el problema resuelta que los pixeles que están en los bordes pierden información ya que el tamaño de la ventana sobresale de la imagen original provocando que queden de menor tamaño a las otras ya que no existe como tal un pixel ahí y así haciendo que en el código aparezca un error. Si la ventana es de tamaño 5x5 y estas en un pixel pegado al borde, todos los pixeles a su izquierda no existirían ya que estarían por fuera de la imagen, y si estas pegado a un borde izquierdo superior, tanto los pixeles de la izquierda como los arriba no existían haciendo que inclusive sea aún menor.

Para solucionar esto es el propio ejercicio te da una pista, crear un borde de pixeles alrededor de la imagen original para tapar con datos esos espacios que sobresalían de la imagen. Dicha función necesita de 6 parámetros principales, la imagen; la cantidad de líneas de pixeles de van arriba, abajo, a la izquierda y a la derecha de la imagen; y la intensidad del que van a tener estos pixeles. Como se nos da ya de por sí un tamaño de la ventana, los valores que van a la derecha y a la izquierda van a ser el ancho dividido 2, lo mismo se aplica para arriba y abajo que es la altura dividida 2. Pero en los tamaños de ventana impares, es necesario restar uno tanto a la altura como a la anchura para que nos quede un valor redondo y no una fracción ya que estos después se redondean solo por Python hacia abajo provocando errores ya que los bordes no son lo suficientemente grandes como para llenar la ventana. La intensidad de la imagen se puede elegir de varias maneras, pero la más conveniente en este caso es hacer que su intensidad sea igual a la de los pixeles cercanos a estos.

Al hacer esto, ahora si funciona el código, pero nos devuelve una imagen con ruido, por lo que es lógico pensar que lo mejor sería aplicar un filtro para reducir ese ruido y tener una imagen lo más limpia posible, por lo que se aplicó después de la ecualización un filtro de mediana con esta intención. Para sorpresa en vez de reducir el ruido se amplifico logrando el caso contrario al buscado, por lo que, tras una investigación,

se determinó que el problema es que la ecualización está resaltando este error que ya de por si existía en la imagen, por lo que, si aplicamos el filtro antes de hacer la ecualización, suavizamos la imagen original y por consecuencia el ruido que proviene de ella, haciendo que el mismo provocado por la ecualización se reduzca lo máximo posible.

Y, por último, al ya tener el código funcionando y devolvieron una imagen con el menor ruido posible, solo queda la cuestión sobre el tamaño que tiene que tener la ventana para destacar y resaltar de la mejor manera los detalles dentro de la imagen. Este tamaño se sacó a partir de prueba y error, probando primero con valores altos y luego yendo hacia los bajos, donde mientras más alto sea el tamaño, más se comporta como una ecualización global, haciendo que los detalles ocultos no se resalten y mientras más bajo, más resalta los detalles ocultos dentro de la imagen comportándose más, como es lógico, como una ecualización local que es lo buscado. Esto no significa que con valores altos no funcione, puede y va a funcionar determinar valores altos considerando que un valor bajo es 5x5 y un valor alto puede ser 50x50. Solo que mientras más alto sea el valor, los detalles ocultos van a resaltar cada vez menos.

PROBLEMA II

Funcionamiento del código

Este código es una implementación de la validación de campos en un formulario escaneado o una imagen de formulario utilizando técnicas de procesamiento de imágenes. El objetivo principal es verificar si los campos en el formulario cumplen con ciertas restricciones en cuanto a la cantidad de caracteres y palabras.

Detalle paso a paso:

- Importa las bibliotecas necesarias: OpenCV (`cv2`) para el procesamiento de imágenes, NumPy (`np`) para operaciones numéricas y Matplotlib (`plt`) para futuras visualizaciones.
- Define una función llamada `contador_celdas` que toma una celda de la imagen (que contiene datos a validar en un renglón del formulario) y devuelve la cantidad de caracteres y palabras en esa celda.
- Umbraliza la imagen para convertirla en una imagen binaria donde las letras estén bien marcadas.
- Convierte la imagen umbralizada a tipo `uint8` para poder usarla en la función `connectedComponentsWithStats`.
- Utiliza `connectedComponentsWithStats` para detectar componentes conectadas en la imagen binaria. Esto ayuda a separar las letras.
- Filtra las estadísticas de las componentes conectadas para eliminar áreas que corresponden a los bordes del campo del formulario. Esto se hace verificando el área de las componentes y manteniendo solo las más pequeñas (que son las letras).
- Ordena las estadísticas en orden ascendente según el valor del eje x para procesarlas en orden.
- Calcula la cantidad de caracteres contando la cantidad de componentes detectadas (que representan las letras).
- Si hay más de un caracter, comienza a contar palabras. Compara la posición de las letras y, si la diferencia en el eje x entre letras consecutivas es mayor que un umbral (8 píxeles en este caso), considera que hay una separación de palabras y suma al contador de palabras.
- Devuelve la cantidad de caracteres y palabras en la celda.
- Define otra función llamada `validacion_formulario` que toma una imagen completa del formulario como entrada.
- Umbraliza la imagen para que las líneas de las columnas y las filas sean de un píxel de ancho.
- Calcula la suma de columnas y filas para encontrar las columnas y filas en la imagen del formulario.
- Utiliza umbrales en las sumas de columnas y filas para detectar las columnas y filas relevantes en la imagen.
- Obtiene los índices de las columnas y filas relevantes.
- Inicializa un diccionario llamado `estadisticas_formulario` para almacenar los resultados de la validación de los campos del formulario.
- Recorre fila por fila en la imagen del formulario (excepto las filas 1 y 6).
- En cada fila, determina a qué campo corresponde (nombre y apellido, edad, correo, etc.) y pasa la celda correspondiente a la función `contador_celdas` para contar caracteres y palabras.
- Compara los resultados con las restricciones específicas para cada campo y registra si el campo cumple o no con las restricciones en el diccionario `estadisticas_formulario`.
- Finalmente, devuelve el diccionario `estadisticas_formulario` que contiene los resultados de la validación de los campos del formulario.
- Lee una imagen en escala de grises del formulario desde el archivo 'formulario_01.png'.
- Llama a la función `validacion_formulario` para validar los campos en la imagen del formulario.
- Imprime el diccionario resultante que contiene el estado de validación de cada campo del formulario.

Ejemplo de ejecución y salida

Luego de la ejecución del algoritmo se mostrará por pantalla el siguiente resultado:

```
(venv) PS E:\UNR\4 - Proc de Imágenes I (IA44)\venv\TP1> & "e:/UNR/4 -
 Proc de Imágenes I (IA44)/venv/Scripts/python.exe" "e:/UNR/4 - Proc d
e Imágenes I (IA44)/venv/TP1/Problema_2.py"
nombre_y_apellido: OK
edad: MAL
mail: OK
legajo: OK
pregunta_1: MAL
pregunta_2: MAL
pregunta_3: MAL
comentarios: OK
(venv) PS E:\UNR\4 - Proc de Imágenes I (IA44)\venv\TP1> & "e:/UNR/4 -
 Proc de Imágenes I (IA44)/venv/Scripts/python.exe" "e:/UNR/4 - Proc d
e Imágenes I (IA44)/venv/TP1/Problema_2.py"
nombre_y_apellido: OK
edad: OK
mail: OK
legajo: MAL
pregunta_1: OK
pregunta_2: OK
pregunta_3: OK
comentarios: OK
(venv) PS E:\UNR\4 - Proc de Imágenes I (IA44)\venv\TP1>
```

```
E:\UNR\4 - Proc de Imágenes I (IA44)\venv>python Problema_2.py
nombre_y_apellido: OK
edad: OK
mail: OK
legajo: MAL
pregunta_1: OK
pregunta_2: OK
pregunta_3: OK
comentarios: OK

E:\UNR\4 - Proc de Imágenes I (IA44)\venv>
```

Nota: El primer ejemplo muestra la ejecución desde VScode con dos imágenes distintas. La segunda captura corresponde a la ejecución desde un terminal.

Planteo de la solución

La idea general de la resolución del problema se pensó en los siguientes pasos:
1. Identificar todos índices de las columnas y de las filas.
2. Ir iterando renglón a renglón (usando los índices de las filas) quedándonos únicamente con la ROI de la celda que contiene la información a validar (es decir haciendo crop de la imagen con índices de las filas y las columnas).
3. Contar la cantidad de caracteres y cantidad de palabras.
4. Validar de acuerdo a las restricciones de dicha celda.

Problemáticas

En cuanto a la comprensión de la lógica de la resolución, no hubo mayores inconvenientes para resolverlo porque nos apoyamos bastante en las ayudas incluidas en el archivo pdf del enunciado.

Las mayores dificultades vinieron de la mano de ser la primera vez que se hace un trabajo de este tipo manipulando matrices, eligiendo umbrales o usando ***cv2.connectedComponentesWithStats()*** pero a medida que aumentábamos la experiencia con la práctica, las dificultades fueron disminuyendo.

REPOSITORIO

GitHub

Todos los archivos correspondientes a la resolución de este Trabajo Práctico se encuentran alojados en un repositorio público en GitHub.

https://github.com/LDemarre/TP1_Procesamiento_de_Imagenes