TECNICATURA UNIVERSITARIA EN INTELIGENCIA ARTIFICIAL (TUIA)

PROCESAMIENTO DE IMÁGENES (PDI)

Docentes de la cátedra:

Dr., Ing. Gonzalo Sad

Ing. Julian Álvarez

Trabajo Práctico 1 (TP1)

Estudiantes (GRUPO 7):

María Celi

Francisco J. Alomar

Bruno Longo

AÑO 2024 (1er CUATRIMESTRE), Rosario.

INTRODUCCIÓN

En el siguiente informe presentamos el proceso de trabajo realizado en el TP1. Al inicio abordamos el problema 1 que requiere la implementación de una ecualización local de histograma para una imagen dada y así descubrir objetos “ocultos”. Luego, tratamos el problema 2, en el que debemos corregir exámenes multiple choice, validar destinos campos (nombre, id, etc.) y devolver una imágen que discrimine los alumnos aprobados de los reprobados. Para ambos problemas hacemos una descripción general del código, mostramos plots para una mayor compresión y describimos problemáticas que surgieron en el proceso. Hacia el final, deducimos algunas conclusiones.

PROBLEMA 1 – Ecualización local de histograma

Durante el cursado aprendimos que el histograma de una imágen es una representación cuantitativa de la distribución de los diferente niveles de intensidad para cada pixel. Es de gran utilidad trabajar con el histograma normalizado y su función de distribución acumulada dado que, al aplicarlo sobre una imágen, se genera una transformación de la intensidad de los píxeles que resulta en un mayor contraste.

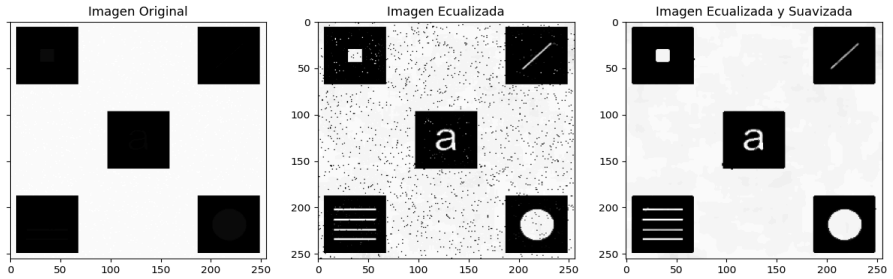
En el presente problema, se propone encontrar objetos “ocultos” en una imagen. Decimos “ocultos” porque casi no son perceptibles a la visión humana, pero al recorrer la imágen como una matriz, por ejemplo, en el primer objeto a descubrir, se aprecia que hay un cuadrado, definido en su mayoría por el valor 10 de intensidad rodeado 0. Es decir, el contraste entre 0 y 10 es casi imperceptible, pero no así para el análisis de PDI. Es por ello que se advierte que un análisis global de ecualización del histograma de la imágen no arrojaría resultados deseables dado que, al aplicar el cálculo utilizando todos los píxeles de la imagen, se perderían la sutiles diferencias en las localidades donde están los objetos a destacar. Ahora bien, mediante la propuesta de resolución de ecualización local de histograma, definimos una ventana de tamaño fijo de píxeles que recorre la imagen original.

Para la resolución del problema generamos el código del archivo TP1_PDI_ej_1.py que sigue los siguientes pasos:

1) Se importan las librerías necesarias.
2) Se define una función llamada local_histogram_equalization() que aplica la función ecualización del histograma local cv2.equalizeHist().
3) En la función local_histogram_equalization():
    - Se obtiene el tamaño de la imagen.
    - Se agrega un borde a la imagen para manejar los píxeles en el borde.

- Se inicializa una imagen de salida como una matriz de ceros con la misma dimensión que la imagen original.
- Se recorre la imagen original y se aplica la ecualización local del histograma en cada ventana definida por el tamaño dado.
- Se devuelve la imagen ecualizada.

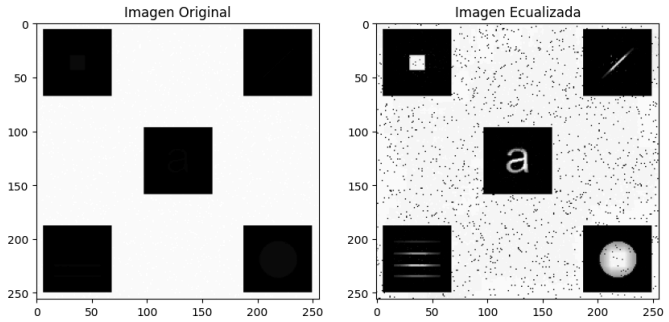
4) Se carga la imagen en escala de grises.
5) Se define un tamaño de ventana para el procesamiento.
6) Se aplica la ecualización local del histograma a la imagen cargada utilizando la función definida anteriormente.
7) Se aplica cv2.medianBlur() con el fin de mejorar el ruido de tipo pepper.
8) Se muestra la imagen original, la imagen ecualizada y suavizada. A continuación mostramos un resultado con una ventada de 20:



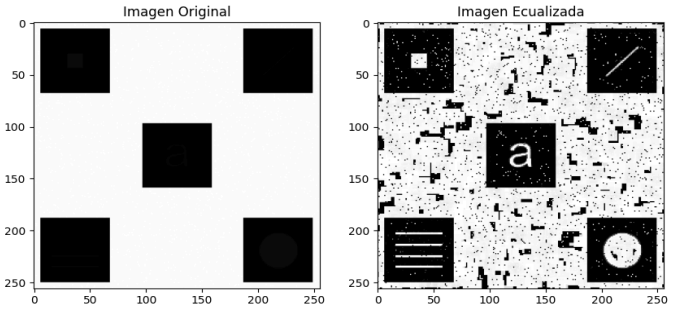
9) Se realizan pruebas adicionales con diferentes tamaños de ventana y se muestran las imágenes original y ecualizada para cada caso.

Una vez que logramos la implementación del algoritmo, nos enfrentamos a dos cuestiones:

1) Nos dimos cuenta que el problema demandaba una exploración del tamaño de la ventana de procesamiento. Es decir, los resultados variaban mucho según sus dimensiones: (a) una ventana de procesamiento muy pequeña detectaba los objetos ocultos, pero generaba una gran cantidad de ruido, (b) mientras que en ventanas de procesamiento muy grandes no generaba ruido, pero tampoco detectaba los objetos de manera adecuada. Comencemos por (b) y mostremos una imágen de tamaño 50 de venta:



Como puede apreciarse, la nitidez de los objetos disminuye al aumentar el tamaño de la venta. Si imaginamos que la ventana de procesamiento tuviera las mismas dimensiones que la imagen original, equivaldría a realizar un análisis global de la imagen. No obstante, si bien disminuye la nitidez de algunos objetos, encontramos uno nuevo en la imágen inferior derecha (un cuadrado dentro del círculo). Ahora, en el caso de (a), mostramos la siguiente imagen con un tamaño de ventana de 10:



Comprendemos que al realizar la ecualización en una región muy reducida de la imagen, el proceso se vuelve más sensible a las pequeñas variaciones de intensidad en el área de la ventana de procesamiento, que, por otra parte, al carecer de una información más global, cada vez que la ventana toma píxeles ya evaluados, vuelve a utilizarlos para una nueva ecualización.

2) Más allá de encontrar una ventana de procesamiento adecuada, siguió apareciendo cierto ruido en la imagen resultante. Este ruido, denominado pepper es deseable que tienda al mínimo aún cuando se cumple con el objetivo de detectar los objetos ocultos. Eliminamos el ruido con una línea de código (paso 7 en la descripción del algoritmo).

Soluciones alternativas

Sumado a la resolución general presentada, hemos desarrollado algunas soluciones

alternativas al problema, que devuelven resultados similares:

1) En el archivo TP1_PDI_ej_1_Contraste exploramos la transformación estrechado de contraste: Esta técnica se utiliza para resaltar detalles finos o sutiles en una imagen al expandir el rango de intensidades de píxeles en ciertas áreas de interés. En este caso resaltamos los objetos variando el valor “E” a conveniencia, obteniendo un resultado óptimo en un E = 0,9.
2) En el archivo TP1_PDI_ej_1_CLAHE.py exploramos la función cv2.createCLAHE() de OpenCV, que se utiliza para crear un objeto CLAHE (Contrast Limited Adaptive Histogram Equalization). El CLAHE es una técnica de procesamiento de imágenes que mejora el contraste local en áreas de una imagen, limitando el exceso de amplificación de contraste en áreas brillantes o oscuras. Los parámetros que toma la función cv2.createCLAHE son:
    - clipLimit: especifica el límite de recorte para la amplificación del histograma.
    - tileGridSize: especifica el tamaño del bloque de mosaico utilizado para calcular la ecualización de histograma adaptativa.

PROBLEMA 2 - Corrección de multiple choice

En este problema se pide corregir automáticamente exámenes múltiple choice. Los exámenes tienen un encabezado con campos para completar datos personales y 25 preguntas con cinco opciones de respuesta cada una. A continuación se enuncian los objetivos de los apartados (a,b,c,d) y los respectivos pasos realizados en el código para lograr su resolución.

Apartado a:

Tomar como entrada la imagen de un examen y mostrar por pantalla cuáles respuestas son correctas y cuáles incorrectas.

1) Se importan las bibliotecas necesarias.
2) Se cargan las imágenes de los exámenes en escala de grises.
3) Se define un diccionario respuestas_correctas_examen que contiene las respuestas correctas para cada pregunta del examen.
4) Se agrupan todas las imágenes de los exámenes en una lista llamada examenes.
5) Hay una función validacion_campos() que devuelve (entre otras variables) el crop de la imágen en booleano de las preguntas y respectivas respuestas sin el encabezado y título.
6) La función respuestas_examen() toma como entrada, para cada exámen, la

imagen del punto 5, y, haciendo uso de la transformada de Hough, encuentra los círculos para cada respuesta, distinguiendo los que han sido rellenados de los que no. La función devuelve un diccionario donde la clave es el número de pregunta (1, 2, 3,..., 25), y el valor es el número de respuesta (1 para A, 2 para B,..., 5 para E) o cero (0) si hay más de una respuesta marcada o ninguna. En la siguiente imágen mostramos un crop, para que se comprenda cómo identifica el algoritmo con Houghs las distintas respuestas para las preguntas de los exámenes. En este caso la respuesta marcada es la A:

7) La función corrección_examen() utiliza el diccionario devuelto por respuestas_examen() y el diccionario definido en el punto (3). En base a esto devuelve un diccionario con las correcciones, el cual tiene como valor el número de pregunta y como clave “OK”, si está bien, o “MAL”, si la respuesta es incorrecta, no se marcó ninguna o más de una.

Apartado b:

Validar los datos del encabezado de la imagen de entrada y mostrar por pantalla el estado de cada campo (Nombre, ID, Código y Fecha) aplicando ciertas restricciones.

1) Se define la función campos_encabezado_y_multiple_choice() que es llamada dentro de la función validacion_campos(), para identificar y segmentar las regiones de nombre, ID, tipo, fecha y opciones de múltiple elección en las imágenes.
2) Se define la función comp_conectados_espacios() para contar el número de caracteres y palabras en un campo de texto mediante la detección de componentes conectados tipo 8.
3) La función campos_encabezado_y_multiple_choice() devuelve los crops en booleano para cada campo a evaluar, de los cuales sólo se tiene el fondo (false, negro, 0), y los componentes conectados. En la siguiente imágen mostramos, un ejemplo para el campo nombre:

4) La validacion_campos() devuelve (entre otras variables) un diccionario donde las claves son los campos a validar, y los valores “OK” o “MAL” de acuerdo a la correspondiente validación.

Apartado c:

Utilizar el algoritmo desarrollado para evaluar las imágenes de exámenes resueltos y reportar los resultados obtenidos.

1) Se realiza un bucle for para iterar sobre cada examen en la lista examenes.
2) Se binariza a booleano cada imagen de examen utilizando un umbral de intensidad (200).
3) Se realizan las validaciones de los campos del encabezado y se corrigen las respuestas del examen para cada examen en el bucle for, en base a lo explicado en los apartados a y b.
4) Se muestra por consola el resultado de la validación de campos y la corrección del examen para cada examen. Para continuar a la siguiente validación y corrección de exámen, es necesario apretar “ENTER” ya así continuar con la ejecución del algoritmo.

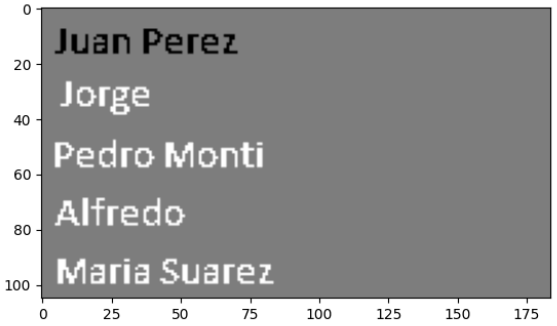
Apartado d:

Generar una imagen de salida que indique qué alumnos han aprobado el examen (con al menos 20 respuestas correctas) y cuáles no. La imagen debe mostrar recortes de los campos de Nombre del encabezado de todos los exámenes y diferenciar de alguna manera aquellos que corresponden a un examen aprobado de uno reprobado.

1) Antes del ciclo definido en el apartado c, se inicializa una variable resultados_concatenados = None, en la cual se irán guardando (concatenando) los array del campo nombre para cada exámen.
2) La función correccion_examen() devuelve (entre otras variables) aprobado, la cual es booleana (True o False) e indica si el alumno aprobó o no, respectivamente.
3) Dentro del ciclo for hay un if en cual se resta o suma la variable nombre a 255, y se guarda en nombre_resultado, generando así un array que tendrá un valor de 255 para lo que en nombre era cero (0, fondo) y de 254 y 256 para los componentes conectados del campo nombre, de acuerdo a si aprobado (True o False), respectivamente.
4) La variable nombre_resultado resultante de cada exámen se agrega a

resultados_contatenados.

8) Se normaliza resultados_concatenados para que el rango de valores sean de 0 a 255 y se convierten a tipo de datos enteros sin signo de 8 bits.
9) Se guarda la imagen de los resultados del examen en un archivo llamado resultados examen.png utilizando cv2.imwrite().
10) Se imprime un mensaje indicando que se ha creado la imagen, se explica el significado de los valores (negro para aprobado, blanco para reprobado) y se muestra la imagen:



La implementación del algoritmo nos presentó algunos desafíos y dificultades que fuimos explorando de manera empírica (es decir, mediante ensayo, prueba y error):

1) Para la validación de los campos e identificar las componentes conectadas utilizamos la ayuda del TP1. La dificultad fue que al aplicar la función indicada, las componentes que debieran ser dos o tres, aparecían como una sola. La solución fue exploratoria, variando el umbral dentro de la función en la siguiente línea de código img_th = examen < 200 . A partir de este umbral (200) el código encontraba la cantidad de componentes que se suponía debía hallar.
2) Determinar los espacios entre componentes para así obtener los caracteres y palabras, fue también un problema con el que nos topamos. La resolución surgió de utilizar la matriz stats que devuelve la función cv2.connectedComponentsWithStats(), que es una que tiene una dimensión de (N, 5), siendo N la cantidad de componentes y 5 la cantidad de columnas, las cuales, para cada componente, en el siguiente orden, define: coordenada x, coordenada y, ancho (pixels), alto (pixels), área(pixel). Con estos datos pudimos hallar la distancia entre las componentes, y, nuevamente, de modo

exploratorio, nos dimos cuenta que cuando adopta un valor mayor o igual a 5 píxeles, significa que la distancia es un espacio. Para lograr esto, no pudimos trabajar sobre la matriz stats tal cual la desempaquetamos de la función. Tuvimos que agregar dos línea de código para ordenarla ind_ord = np.argsort(stats[:,0]) stats_ord = stats[ind_ord] y poder operar sobre ella.

CONCLUSIONES

Después de revisar el trabajo, nos hemos enfocado en las conclusiones derivadas del proceso. El procesamiento de imágenes se presenta como un desafío inicial debido a las complejas operaciones matriciales. Aunque habíamos estudiado estas operaciones en teoría, nos enfrentamos a ellas por primera vez en la práctica. En la resolución de problemas, hemos adoptado un enfoque principalmente exploratorio, basado en ensayo y error. Esto nos ha llevado a reconocer la importancia del análisis individual en cada caso, a pesar de que existen soluciones predefinidas para algunos problemas específicos.