PROCESAMIENTO DIGITAL DE IMÁGENES I

Trabajo Práctico Nº1

Año 2023

Mateo Gravi Fiorino,

Alejo Lo Menzo,

Tomás Navarro Miñon,

Ramiro Sagrera.

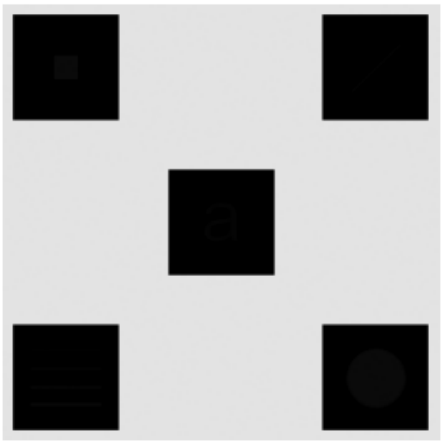
Problema 1. – Ecualización local de histograma

Problema 2 – Validación de formulario

Repositorio de GitHub: https://github.com/rama811/tp1_pdi

Problema 1.

Para el primer problema, se nos presenta una imágen a la cuál debemos aplicarle la técnica de ecualización del histograma. Para hacerlo, definimos una ventana cuadrada o rectangular y se desplaza pixel a pixel. Esa información la usamos para ecualizar localmente el histograma. Desarrollamos una función que toma como entrada la imagen y el tamaño de la ventana. Luego podremos ver los detalles ocultos en la imágen.


Imágen de entrada.

Primero definimos la función

```python
def histograma_ecualizar(image, window_size):

    alto, ancho = image.shape

    # Calcula la mitad del tamaño de la ventana
    tamanio_vent = window_size // 2

    # Crea una copia de la imagen de salida
    imagen_salida = image.copy()

    for y in range(tamanio_vent, alto - tamanio_vent):
        for x in range(tamanio_vent, ancho - tamanio_vent):
            # Extrae la ventana local
            window = image[y - tamanio_vent:y + tamanio_vent + 1, x - tamanio_vent:x + tamanio_vent + 1]

            # Calcula el histograma local
            hist = cv2.calcHist([window], [0], None, [256], [0, 256])

            # Aplica la ecualización del histograma local
            ventana = cv2.equalizeHist(window)

            # Coloca los píxeles ecualizados en la imagen de salida
            imagen_salida[y, x] = ventana[tamanio_vent, tamanio_vent]

    return imagen_salida
```

Y podemos ver paso por paso cómo funciona:

alto, ancho = image.shape:

Obtiene las dimensiones de la imagen de entrada.

tamanio_vent = window_size // 2:

Esta línea calcula la mitad del tamaño de la ventana. La razón por la que se hace esto es para determinar el rango de píxeles que se tomarán en cuenta alrededor de un píxel específico en la imagen. La ventana se desliza a través de la imagen y se centra en cada píxel. Al calcular la mitad del tamaño de la ventana, se obtiene el número de píxeles hacia arriba, abajo, a la izquierda y a la derecha del píxel central. Esta mitad se utiliza para asegurar que la ventana esté centrada correctamente en cada píxel y no se salga de los límites de la imagen.

imagen_salida = image.copy():
Crea una copia de la imagen de entrada para almacenar la imagen de salida.

for y in range(tamanio_vent, alto - tamanio_vent):
Itera a través de las filas de la imagen, excluyendo los bordes según el tamaño de la ventana.

for x in range(tamanio_vent, ancho - tamanio_vent):
Itera a través de las columnas de la imagen, excluyendo los bordes según el tamaño de la ventana.

window = image[y - tamanio_vent:y + tamanio_vent + 1, x - tamanio_vent:x + tamanio_vent + 1]:

Esta línea extrae una ventana de píxeles alrededor del píxel central en la posición (y, x) de la imagen. El rango de índices utilizado (por ejemplo, y - tamanio_vent hasta y + tamanio_vent + 1) se asegura de que se tomen los píxeles correctos dentro de la ventana centrada en el píxel actual.

hist = cv2.calcHist([window], [0], None, [256], [0, 256]):

Calcula el histograma de intensidad de la ventana local. Un histograma de intensidad muestra la distribución de los niveles de gris en una imagen o una región de la imagen. En este caso, se calcula el histograma de la ventana para entender cómo están distribuidos los niveles de gris en esa área específica.

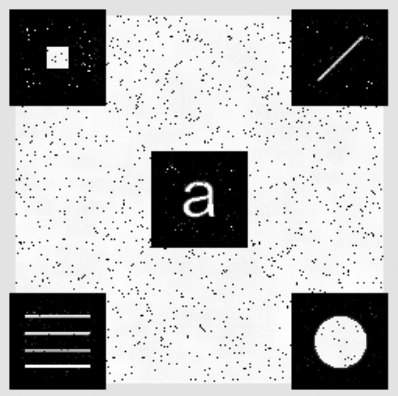
ventana = cv2.equalizeHist(window):

Aplica la ecualización del histograma a la ventana local. La ecualización del histograma es un método para mejorar el contraste en una imagen al redistribuir los niveles de gris de manera uniforme a lo largo de todo el rango. Aplicar esto a la ventana local mejora el contraste en esa área específica de la imagen.

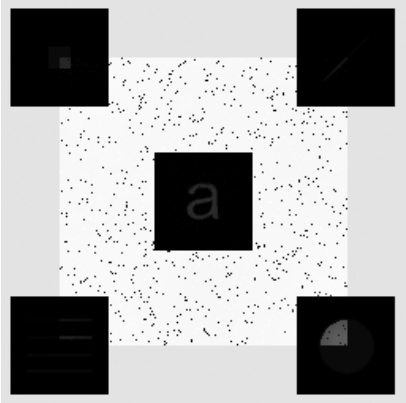
imagen_salida[y, x] = ventana[tamanio_vent, tamanio_vent]:

Coloca los píxeles ecualizados de la ventana de vuelta en la imagen de salida. tamanio_vent se utiliza para obtener el píxel central de la ventana ecualizada y se coloca en la misma posición (y, x) en la imagen de salida.

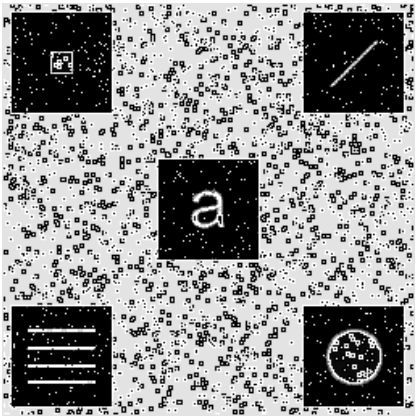
Nos dimos cuenta que según el tamaño de la ventana obtenemos resultados diferentes. Por ejemplo, con una ventana de tamaño 20, podemos distinguir los elementos ocultos.



Con una ventana de tamaño 75:



Con tamaño 5:



Problema 2.

El problema planteado implica validar campos de un formulario basándose en una imagen del mismo. El formulario tiene varios campos, cada uno con restricciones específicas:

1. **Nombre y Apellido:** Debe tener 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:** Debe tener 8 caracteres formando 1 sola palabra.
5. **Preguntas**: Cada una de las 3 preguntas debe marcarse con un solo caracter, ya sea "SI" o "NO". No pueden estar ambas vacías ni ambas completas.
6. **Comentarios:** No debe contener más de 25 caracteres.

Para abordar este problema, se propone un algoritmo que implica varios pasos. Primero, se umbraliza la imagen para detectar las líneas verticales y horizontales que dividen el formulario en filas y columnas. Luego, se utilizan estos datos para identificar las celdas del formulario. Dentro de cada celda, se aplican técnicas de procesamiento de imágenes, como la detección de componentes conectadas, para identificar los caracteres escritos en el formulario.

Es importante manejar los posibles errores, como los píxeles de las líneas divisorias que podrían quedar dentro de las celdas. Para evitar esto, se eliminan las componentes conectadas de área muy pequeña.

El algoritmo devuelve el estado de cada campo del formulario (OK o MAL) después de realizar todas las validaciones requeridas. Se proporcionan pistas sobre cómo abordar la detección de líneas divisorias y cómo evitar problemas con los píxeles no deseados dentro de las celdas.

El proceso implica una serie de pasos cuidadosos para segmentar la imagen, detectar las celdas y validar el contenido de cada celda según las restricciones dadas. Este enfoque garantiza la integridad de los datos ingresados en el formulario.

A continuación vemos el código:

Tiene como objetivo validar los campos de un formulario basándose en una imagen del formulario proporcionada como matriz. Utiliza las bibliotecas numpy para manipulación de matrices, cv2 (OpenCV) para procesamiento de imágenes y matplotlib.pyplot para visualización.

Primero, se define la función contador_celdas(image) que toma una celda de la imagen del formulario y devuelve la cantidad de caracteres y palabras en esa celda. La función utiliza umbralización para convertir la imagen en binaria y detecta componentes conectados para identificar letras, guiones o barras. Luego, se filtran los elementos para eliminar los bordes del campo y se cuentan las letras y palabras en la celda.

Luego, la función validacion_formulario(image) procesa la imagen del formulario completa. Primero, umbraliza la imagen y encuentra las columnas y filas externas del formulario. Luego, recorre cada fila del formulario (excepto las filas 1 y 6) y utiliza la función contador_celdas() para validar cada campo. Las validaciones se hacen de acuerdo con ciertas restricciones específicas para cada campo, como el número máximo de caracteres y palabras permitidas en un campo.

Finalmente, se devuelve un diccionario llamado estadisticas_formulario que contiene el resultado de las validaciones para cada campo del formulario.

Está diseñado para procesar un formulario específico cuya ruta de archivo se proporciona en la variable img. Después de procesar la imagen del formulario, imprime el resultado de la validación para cada campo. Cabe mencionar que la ruta del archivo de la imagen del formulario debe ser ajustada para que el código funcione correctamente en un entorno específico.