Pasos Previos

Para comenzar el trabajo práctico, descargamos todo el material proporcionado por la cátedra, disponible en el repositorio de Git indicado.

Todos los elementos fueron almacenados dentro de una misma carpeta, la cual incluye el script **“Tp1_PDI”**, encargado de resolver los problemas planteados.

Script

1. **Importación de librerías:**
   Se importaron las funciones necesarias para el procesamiento de imágenes y la visualización de resultados.

2. **Carga y preparación de las imágenes:**
   Se descargaron los distintos formularios y la imagen con detalles escondidos. Luego, todas las imágenes fueron convertidas a **escala de grises** para facilitar su procesamiento.

3. **Desarrollo de los problemas:**

   Para resolver el primer problema (detalles escondidos), investigamos diferentes métodos de mejora de contraste y seleccionamos la técnica más eficiente.
   Creamos una función denominada **`ecualizacion_local`**, que recibe como parámetros la imagen a procesar y el tamaño de la ventana.
   Dentro de esta función utilizamos un objeto de OpenCV llamado **CLAHE**, que realiza la ecualización de histograma de forma local.

   - El parámetro **`clipLimit=2.0`** controla la amplificación del contraste local. Un valor alto puede generar ruido excesivo, mientras que un valor bajo apenas produce cambios.

   - El parámetro **`tileGridSize`** divide la imagen en bloques según el tamaño de ventana especificado.

   Finalmente, aplicamos esta ecualización sobre la imagen y graficamos tanto la imagen resultante como sus histogramas para analizar los efectos del proceso.

   Para resolver el segundo problema decidimos estructurar el código en funciones modulares, comenzando por la **segmentación de los formularios**.

   Implementamos la función **`extraer_celdas_de_imagen`**, que recibe como parámetro principal la imagen del formulario, junto con los siguientes valores configurables:

- **`bin_thresh=150`**: umbral utilizado para binarizar la imagen.

- **`row_thresh_offset=80`**: umbral empleado para detectar las **líneas horizontales** del formulario.

- **`col_thresh_offset=350`**: umbral utilizado para detectar las **líneas verticales**.

- **`min_row_height=3`**: parámetro que define el **alto mínimo** para considerar un renglón como válido, evitando detectar separaciones pequeñas o ruido.

- **`special_row_start=6`** y **`special_row_end=9`**: algunos renglones del formulario presentan **tres divisiones verticales** en lugar de dos, por lo que estos parámetros indican en qué filas se debe aplicar un **doble recorte** adicional.

La función trabaja sobre la matriz binaria (de valores `True`/`False`) obtenida tras el umbralado. En primer lugar, identifica las **coordenadas de las líneas horizontales y verticales** a partir de las proyecciones de píxeles. Luego, utiliza esas coordenadas para **recortar los renglones** y, dentro de cada uno, **dividir las columnas** correspondientes, generando así las distintas celdas del formulario.

Al finalizar la ejecución, la función devuelve una **lista de listas**, donde cada elemento representa un renglón y contiene las **celdas individuales** que lo componen.

El principal desafío en esta etapa fue ajustar correctamente los valores de los umbrales, ya que pequeñas variaciones en estos parámetros podían alterar la detección de líneas y, en consecuencia, afectar la segmentación final del formulario.

Una vez finalizada la función de extracción de celdas, se llevó adelante la función **seleccionar_celdas.** Esta función trabaja sobre las celdas que extrae la función anterior y devuelve un diccionario con los pares clave-valor, donde la clave es el nombre representativo de la celda y el valor es el recorte de dicha celda. De esta forma se reduce el conjunto a solamente las celdas que deberán ser validadas.

Luego se desarrollaron una función para cada validación:

- **validar_nombre()**
- **validar_edad()**
- **validar_mail()**
- **validar_preguntas()**
- **validar_comentario()**

Estas funciones son similares en cuanto a estructura. Todas reciben el crop de la celda correspondiente, se transforma de binaria a (0-255) y con el método **cv2.connectedComponentsWithStats()** se reconocen los objetos de la imagen. Se trabaja sobre la componente **stats** que es la que aporta la información de los objetos reconocidos. Se ignora el fondo con un slicing y las líneas verticales que quedaron

en el crop filtrando por la altura de cada objeto reconocido (la línea tiene el alto de la imagen).

Luego se agrega cada ‘mancha’ a una lista final, y se realiza la validación correspondiente de cada celda. Por ejemplo, en el nombre, corroborar que haya máximo 25 caracteres y mínimo 2 palabras. Esto se valida evaluando el espacio entre cada ‘mancha’, consideramos que si la distancia es mayor a 5 pixeles es un espacio. Por último se devuelve un print en cada caso correspondiente si pasa la validación o no, y luego retorna True o False, según se haya completado la celda de forma correcta o incorrecta.

La función principal simplemente recibe la imagen de un formulario, el nombre (string) y el ID (número entero), crea el diccionario llamando a la función **seleccionar_celdas(),** asigna el crop de cada celda a una variable y llama a cada función de validación con su respectiva celda. También crea una lista con el resultado de cada validación, es decir True o False, según si la celda pasa la validación o no, y al final se chequea si el formulario está bien o mal completado. También se crea una lista con el ID del formulario y si está OK o MAL cada casillero, en un bucle for a partir de la lista con los valores booleanos.

Finalmente la función principal retorna la imagen del crop del nombre y apellido (para luego ser utilizado en el apartado de la imagen con las celdas de los nombres), un booleano que indica si el formulario está bien o mal en su totalidad, y las filas con las validaciones de cada celda para el archivo csv.

En el problema 2 B, se realizó una función llamada **validar_por_tipo()** que no recibe argumentos, itera sobre 2 listas de tuplas, una para cada tipo de formulario (A o B) y muestra todas las validaciones aclarando el nombre y tipo de formulario.

Para el problema 2 C, se creó una lista vacía de las imágenes. Luego en un bucle for se asigna cada salida de la función principal para cada formulario, y se guarda la imagen en la lista. Luego se crean los bordes de color para agregar al crop del nombre y apellido de la persona que haya completado bien o mal el formulario, según corresponda. En un bucle for que itera sobre la lista de imágenes (tuplas imagen, booleano si es válido o no), se convierten los crops de las imágenes a BGR, se le agrega el recuadro del color correspondiente y se guardan las imágenes en una lista final. Luego se calculan las dimensiones del lienzo, se crea y por último con un bucle que itera sobre las imágenes listas, se pegan sobre el lienzo. Finalmente, se guarda la imagen resultante en un archivo llamado **resumen_final.jpg.**

Para el problema 2 D, se creó una lista vacía donde iremos guardando las filas en formato csv. Luego, con un bucle for capturamos los 3 valores que nos devuelve la función principal y agregamos la fila con los valores csv a la lista que creamos anteriormente. Después creamos una lista llamada **‘csv_header’**, la cual contendrá los encabezados de las columnas de nuestro archivo. Por último, creamos el archivo **‘resultados_validacion.csv’** donde agregaremos el encabezado y las filas de las validaciones de nuestros formularios.

Resultados Obtenidos

Problema 1 – Imagen con detalles escondidos

- Recuadro superior izquierdo: **cuadrado**
- Recuadro superior derecho: **/**
- Recuadro central: **letra “a”**
- Recuadro inferior izquierdo: **líneas horizontales**
- Recuadro inferior derecho: **círculo**

A medida que aumenta el tamaño de la ventana, los detalles ocultos en los recuadros negros comienzan a apreciarse con mayor claridad.
Sin embargo, cuando el tamaño de la ventana alcanza aproximadamente **80**, la imagen empieza a perder definición: los contornos se difuminan y los colores comienzan a mezclarse. El fondo deja de presentarse como una superficie gris uniforme y muestra variaciones en sus tonalidades, mientras que los recuadros negros reducen su intensidad y contraste.

Problema 2 – Validación de formularios

Como resultado del algoritmo aplicado a los formularios, se obtienen cuales son los campos son correctos y cuales son incorrectos, y si el formulario en su totalidad está completo correctamente o no. Para cada apartado de este problema cambia el formato en el cual se observa la validación de los formularios, ya sea para uno en particular (2 A) o para los cinco formularios de manera cíclica (2 B). En el apartado C se obtienen solamente los nombres de las personas que completaron cada formulario, y un indicador según si lo completó bien o mal. Por último en el apartado D, se obtiene un archivo .csv con la validación de cada celda del formulario.