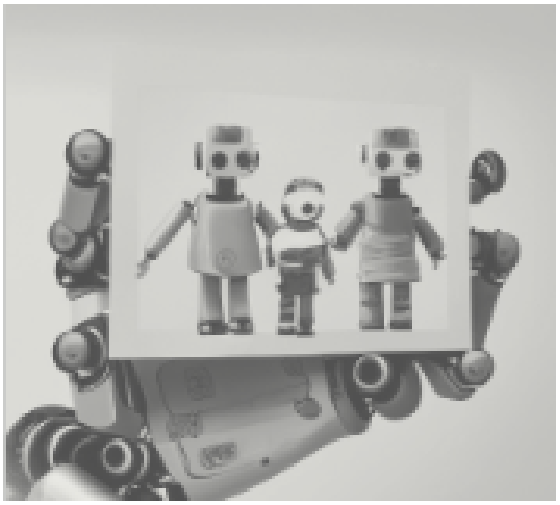
TRABAJO PRÁCTICO 1

IA 4.4 Procesamiento de Imágenes I

Tecnicatura Universitaria en Inteligencia Artificial



Integrantes:

- Florencia Fernández
- Lionel Palermo
- Leandro Salvañá

02/10/2023

Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

En el contexto de aprendizaje y abordaje de problemas planteados en la asignatura Procesamiento de Imágenes I y habiéndose dictado las Unidades 1, 2 y 3 pertinentes al abordaje de las temáticas Fundamentos, Transformación y Filtrado y, Detección de Bordes y Segmentación, se realiza el informe correspondiente con la finalidad de detallar la resolución encontrada, su idea y desarrollo, como así también dar a conocer las dificultades o desafíos hallados en el camino. Las aclaraciones se realizan a continuación, una para cada ejercicio, en el mismo órden en que estos fueron planteados por los docentes.

1 - Ejercicio 1

1.1 - Se solicita como resolución

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 cuáles son los detalles escondidos en las diferentes zonas de la misma. Analice la influencia del tamaño de la ventana en los resultados obtenidos.

1.2 - Enfoque preliminar

Recordando la teoría vista correspondiente a la unidad de Transformación y Filtrado y, considerando lo indicado como resolución, primeramente se plantea una idea general de resolución: se busca revelar detalles ocultos en la imagen logrado por la implementación de ecualización local del histograma que devolvería una imagen con mayor contraste. Se debe realizar por un tamaño de ventana específico. Por lo tanto, en el desarrollo de la función se precisa:

- Contar con las librerías necesarias para la resolución: numpy, cv2, matplotlib.
- Contar con el tamaño de ventana y con la imagen original como parámetros de la función.
- Conocer las dimensiones de la imagen original para crear una copia a la que se vayan asignando los valores del píxel central de la porción de imagen a la que se aplica la ecualización local del histograma.
- Hallar una forma de recorrer la imagen original para aplicar la ecualización local del histograma. Posiblemente con bucles for.
- Calcular el histograma de los puntos dentro de la ventana y obtener de esta manera, una transformación local de ecualización del histograma y utilizarla para mapear el nivel de intensidad del pixel centrado en la ventana bajo análisis, extrayendo así el valor del píxel correspondiente a la imagen procesada.
- Asignar el valor a la imagen de salida y, una vez completada, retornarla.

Previo a encarar la escritura del código se realizan conjeturas sobre el resultado esperado: Una vez aplicada la ecualización local del histograma, se espera que en la imagen de ejemplo compartida se revelen en el área de la misma tanto ruido como datos de interés. Los detalles hallados dentro de los cuadrados negros de la imagen se encontrarán en

valores de intensidad más alta, mientras que los hallados dentro de los cuadrados blancos tendran valores de intensidad más bajos, relativo al cuadrado donde fueron detectados.

1.3 - Desarrollo

El desarrollo de la resolución se llevó a cabo siguiendo el enfoque preliminar mencionado anteriormente. Se implementó una función en Python que toma como entrada la imagen y el tamaño de la ventana, y devuelve la imagen con la ecualización local del histograma aplicada.

Se utilizó la biblioteca OpenCV (cv2) para operaciones de procesamiento de imágenes y NumPy para manipulaciones de matrices. Además, se hizo uso de la función cv2.copyMakeBorder para agregar un borde reflectante alrededor de la imagen, lo que facilitó el procesamiento en los bordes.

Además, se realizaron una serie de validaciones pertinentes sobre los argumentos de entrada relacionados al tipo de dato y las dimensiones requeridas para su validez, arrojando errores descriptivos en caso de no cumplirlas.

Estructura de la función: Archivo `funciones.py`

```
import cv2
```

```
import numpy as np

import matplotlib.pyplot as plt

# Ejercico 1

## Función de ecualización del histograma local

def local_hist_eq(img, window_size=[3, 3]):

    """

    Aplica ecualización local del histograma a una imagen en escala de grises.



    :param img: Imagen en escala de grises representada como un arreglo numpy.

    :param window_size: Tamaño de la ventana para la ecualización local del
histograma (por defecto, [3, 3]).

    :return: Imagen con ecualización local del histograma aplicada.
```

```
    """

    # Se valida que la imagen ingresada sea un arreglo numpy con forma (x, y)

    if not isinstance(img, np.ndarray) or img.ndim != 2:

        print(img.ndim)

        raise ValueError("La imagen debe ser un arreglo NumPy en escala de grises con forma (x, y).")

    # Se valida que el window_size ingresado sea una lista de la forma [a, b] donde a y b son números enteros

    if not isinstance(window_size, list) or len(window_size) != 2 or not all(isinstance(val, int) for val in window_size):

        raise ValueError("window_size debe ser una lista de dos números enteros [a, b].")

    # Se obtienen las dimensiones de la ventana y se comprueba que sean válidas

    window_width, window_height = window_size

    if window_width <= 1 or window_height <= 1 or window_width > img.shape[0] or window_height > img.shape[1]:

        raise ValueError("Los valores en window_size deben ser enteros mayores que 1 y menores que las dimensiones de la imagen.")

    # Se obtienen las dimensiones de la imagen ingresada

    height, width = img.shape

    # Se crea la imagen de salida inicialmente vacía del mismo tamaño que la imagen ingresada

    img_output = np.zeros((height, width), dtype=np.uint8)

    # Se calcula la mitad del ancho y alto de la ventana (vecindario) a utilizar para la ecualización local
```

```
    half_window_width = window_width // 2

    half_window_height = window_height // 2

    # Se crea un padding en la imagen original utilizando cv2.BORDER_REFLECT

    img_padded = cv2.copyMakeBorder(img, half_window_height,
half_window_height, half_window_width, half_window_width, cv2.BORDER_REFLECT)

    # Bucle for para recorrer el alto de la imagen

    for y in range(height):

        # Bucle for para recorrer el ancho de la imagen

        for x in range(width):

            # 1) Se crea un recorte del tamaño de la ventana solicitada

            window = img_padded[y:y + window_height, x:x + window_width]

            # 2) Se aplica ecualización por el histograma (cv2.equalizeHist) a
dicho recorte

            hist_eq_window = cv2.equalizeHist(window)

            # 3) Se obtiene el valor de intensidad del pixel central del
recorte ecualizado

            central_pixel_value = hist_eq_window[half_window_height,
half_window_width]

            # 4) Finalmente, se asigna ése valor en el pixel (y, x) de la
imagen de salida "img_output"

            img_output[y, x] = central_pixel_value

    return img_output
```

1. Validación de la entrada: Se aseguró que la imagen de entrada sea un arreglo NumPy en escala de grises y que el tamaño de la ventana sea una lista de dos

números enteros mayores o iguales a 1 y que no superen las dimensiones de la imagen original.
2. Creación de la imagen de salida: Se generó una imagen vacía del mismo tamaño que la imagen original para almacenar los resultados.
3. Definición de la ventana: Se estableció el tamaño de la ventana que se desplazaría sobre la imagen para realizar la ecualización local. Esto permitiría analizar regiones específicas en lugar de toda la imagen a la vez.
4. Adición de bordes: se agregan bordes a la imagen original utilizando como parámetros la mitad del alto y ancho de la ventana.
5. Procesamiento local: Se implementó un bucle que recorre la imagen original. En cada iteración, se extrae una ventana del tamaño especificado alrededor del píxel actual.
6. Ecualización del histograma local: Se aplicó la ecualización del histograma a la ventana obtenida en el paso anterior.
7. Asignación del valor resultante: El valor de intensidad del píxel central de la ventana ecualizada se asignó al píxel correspondiente en la imagen de salida.
8. Repetición: Se repitió el proceso hasta haber procesado toda la imagen.
9. Devolución imagen: Se retornó la imagen de salida con la ecualización local del histograma implementada.

Aplicación de la función: Archivo `ejercicio1.py`

En el archivo de Python mencionado arriba, se importa la función del ejercicio 1 y se hace uso de ella sobre la imagen de ejemplo, usando distintos valores de ventana y mostrando los resultados para cada uno de ellos junto con la imagen original.

```
import cv2

import matplotlib.pyplot as plt

from funciones import local_hist_eq



## Ecualización local del histograma ##

# Ruta de la imagen

image_path = './content/Imagen_con_detalles_escondidos.tif'



# Se carga la imagen TIFF

img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
# Se definen los tamaños de ventana para realizar la ecualización local del histograma

window_sizes = [[2, 2], [5, 5], [10, 10], [15, 15], [20, 20],  [25, 30], [40, 40], [60, 60]]



# Se crea una figura que contendrá todas las imágenes

fig, axes = plt.subplots(3, 3, figsize=(15, 5))



# Se agrega la imagen original a la figura

row = 0

column = 0

axes[row,column].imshow(img, cmap='gray', vmin=0, vmax=255)

axes[row,column].set_title(f"Imagen original")

axes[row,column].axis('off')



# Se aplica la ecualización local del histograma para cada tamaño de ventana y se muestran las imágenes en subgráficos

for i, window_size in enumerate(window_sizes):

    output_image = local_hist_eq(img, window_size)



    # Se definen las filas y columnas para la imagen

    column = column + 1

    if i == 2 or i == 5:
```

```
        row = row + 1

        column = 0



    # Se muestra la imagen resultante en el subgráfico correspondiente

    axes[row, column].imshow(output_image, cmap='gray', vmin=0, vmax=255)

    axes[row, column].set_title(f"Tamaño de ventana:
{window_size[0]}x{window_size[1]}")

    axes[row, column].axis('off')



plt.show()
```

1.4 - Conclusiones técnicas

Exposición de detalles hallados:

Se han detectado elementos de interés, es decir, considerados coherentes por poder asociarse a representaciones de objetos o elementos conocidos y con un significado y no una forma definida de manera irregular, contenidos en los cuadrados de baja intensidad ubicados en las esquinas y en el centro de la imagen original.

- En el cuadrado superior izquierdo se halló contenido un cuadrado de pequeñas dimensiones en relación a su contenedor, pero de suficiente tamaño para ser divisado como tal.
- En el cuadrado superior derecho se halló una línea fina diagonal inclinada a 45°.
- En el cuadrado ubicado en el centro se halló la letra “a” minúscula del alfabeto.
- En el cuadrado inferior izquierdo se hallaron 4 líneas horizontales contenidas en el mismo.
- En el cuadrado inferior derecho se halló un círculo de diámetro cercano a la arista de su cuadrado contenedor y centrado en el mismo.

En toda la extensión de la imagen se detectaron segmentos de ruido, considerados como tal por su distribución aleatoria y aparentemente desprovista de significado.

Impacto del tamaño de la ventana:

- Nivel de detalle resaltado:

  Ventana Pequeña: Si la ventana es pequeña, se resaltan detalles finos y de pequeña escala en la imagen. Esto puede ser útil para realzar texturas y estructuras muy detalladas.

  Ventana Grande: Aplicando una ventana grande parece ser más efectiva para resaltar características de mayor escala, ya que considera un área más amplia en cada cálculo de ecualización local. Esto puede ser útil para realzar características más grandes y suaves.

- Sensibilidad a ruido y variaciones locales:

  Ventana Pequeña: Usando una ventana pequeña se observa más sensibilidad al ruido y a las variaciones locales. Esto puede llevar a una amplificación del ruido, notando cómo se reduce a medida que se aumenta el tamaño de la ventana.

  Ventana Grande: Utilizando una ventana grande se observa que puede ayudar a suavizar variaciones locales y reducir la influencia del ruido, ya que promedia información de una región más extensa.

- Posibilidad de Pérdida de Detalles Finos o Ruido:

  Ventana Grande: Observando el uso de una ventana muy grande se destaca que puede suavizar demasiado la imagen, lo que podría resultar en la pérdida de detalles finos e importantes.

  Ventana Pequeña: Al hacer uso de una ventana muy pequeña se examina que puede llevar a un realce excesivo de detalles locales, incluido el ruido.

- Tiempo de Procesamiento:

  Si bien no se han utilizado ventanas excesivamente grandes y el tiempo de procesamiento no ha sido extenso, se intuye que procesar una ventana pequeña es más rápido en términos de tiempo de cómputo, ya que involucra menos píxeles en cada cálculo y utilizar una ventana grande podría aumentar el tiempo de procesamiento, ya que implica un mayor número de píxeles en cada iteración.

Se experimentó con diferentes tamaños de ventana y se evaluaron los resultados para determinar la configuración óptima para la naturaleza específica de la imagen bajo observación.

En base a lo detallado anteriormente y por observación empírica se determina que un tamaño de ventana de 20x20 pixeles es el adecuado para detectar los elementos significativos en la imagen analizada.

Tamaños de ventana menores muestran imágenes menos nítidas para las áreas de los detalles de interés y más ruido y, ventanas mayores muestran menos ruido pero suavizan los elementos detectados en los cuadrados de las esquinas y el centro de la imagen.

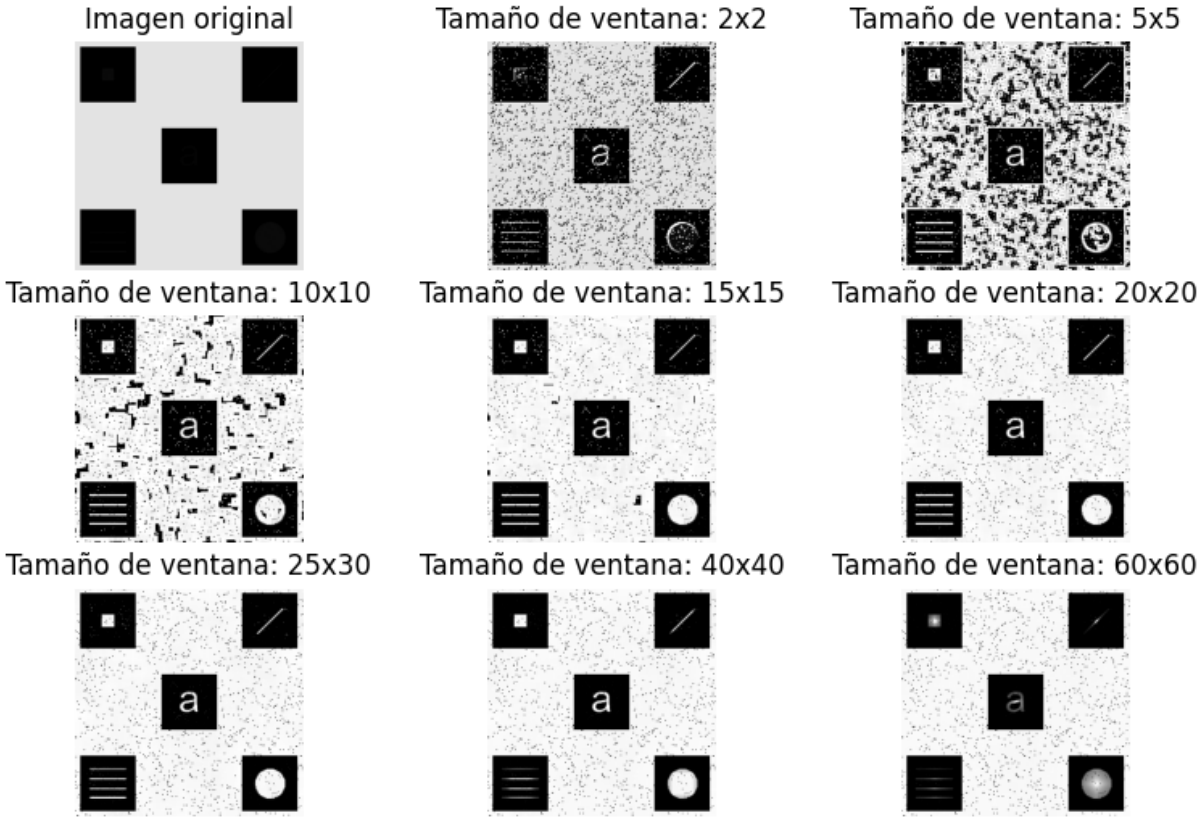


Fig. 1.1 Se muestra la imagen original y los resultados de aplicar la ecualización local del histograma sobre la misma para distintos tamaños de ventana.

1.5 - Desafíos encontrados

Para la resolución de este problema y, dado que fue el primero, principalmente el desafío estuvo en el conocimiento y manejo de las herramientas utilizadas. Se tomó un tiempo previo a investigarlas para hacer uso de las adecuadas para el problema concreto.

Además había que asegurarse de que la ventana se mantuviera dentro de los límites de la imagen para evitar errores de índices fuera de rango. Lo que se contempló con la creación de un padding en la imagen original.También se tuvo que considerar la forma en que se realiza la ecualización del histograma en cada ventana y la asignación de valores resultantes.

1.6 - Reflexiones sobre el abordaje

La metodología empleada nos permitió abordar el problema de manera sistemática y modular. Dividir el proceso en pasos claros, nos facilitó la implementación y la depuración

del código. Además, se buscó hacer un uso efectivo de las bibliotecas de Python específicas para procesamiento de imágenes, para contribuir a la eficacia del desarrollo. Sin embargo, es importante tener en cuenta que esta implementación podría necesitar ajustes o mejoras según los requisitos específicos de diferentes situaciones o tipos de imágenes.

2 - Ejercicio 2

2.1 - Se solicita como resolución

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.

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.

2.2 - Enfoque preliminar

En el planteamiento inicial de la resolución se buscó detectar los pasos necesarios para llegar al resultado solicitado y modularizarlos ya que cada uno podía considerarse un subproblema a resolver para integrarlo luego en la solución final.

Por lo tanto, para resolver el problema nos basamos en una serie de pasos:

1. Segmentación de celdas: Primero, se debía identificar y segmentar las celdas que contienen la información en el formulario. Para esto, se utilizó una técnica de umbralización para destacar los contornos de las celdas.

2. Extracción de características: Luego, se necesitaba extraer características específicas de cada celda, como el número de caracteres y el número de palabras. Esto es crucial para verificar si los campos cumplen con los requisitos establecidos.

3. Validación de campos: A partir de las características extraídas, se podía proceder a validar cada campo según las restricciones proporcionadas. Por ejemplo, verificar la longitud del nombre y apellido, la cantidad de caracteres en la edad, etc.

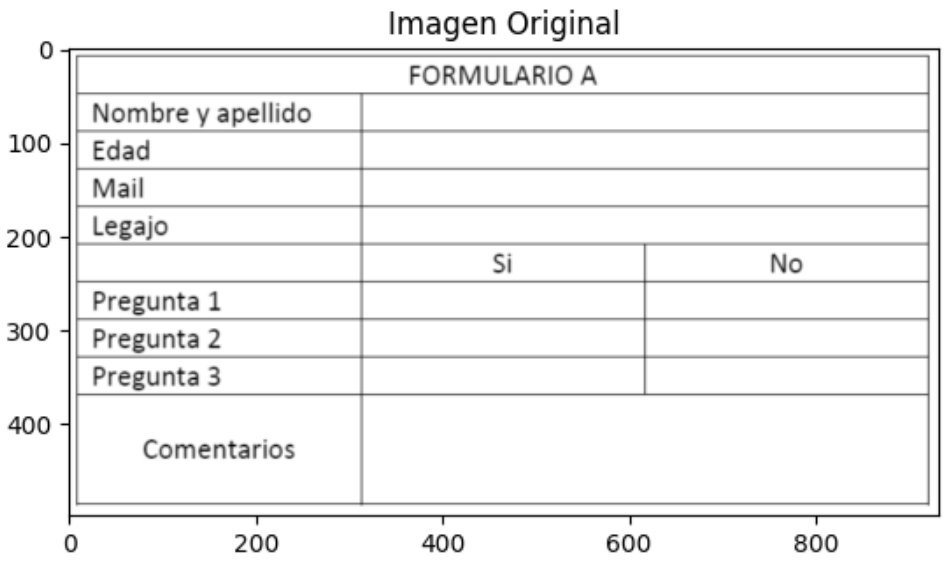
4. Generación de informe: Finalmente, se debía compilar los resultados de la validación en un informe legible que muestre el estado de cada campo.

2.3 - Desarrollo

En primer lugar, se llevó a cabo el preprocesamiento de la imagen mediante la umbralización de la imagen del formulario importada. Para ello, se utilizó la función "threshold" de Open Computer Vision (OpenCV) en la siguiente implementación:

```
umbral = 200

img_umbralizada = cv2.threshold(img_blank, umbral, 255,
cv2.THRESH_BINARY)[1]
```

El resultado es el siguiente



Para abordar el problema relacionado con la variabilidad de los bordes exteriores en todos los formularios, se procedió a invertir los colores de la imagen para que los bordes y los caracteres quedaran en blanco sobre un fondo negro. Posteriormente, se empleó la función "findContours" de OpenCV para detectar los contornos del formulario. Estos contornos se ordenaron de manera descendente según su área, y se seleccionó el contorno con la mayor área, correspondiente al borde exterior del formulario. La implementación fue la siguiente:

```
# Invertir los colores (si los bordes están en negro)

img_umbralizada_invertida = cv2.bitwise_not(img_umbralizada)



# Encontrar contornos en la imagen umbralizada invertida
```

```
contornos, _ = cv2.findContours(img_umbralizada_invertida,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)



# Ordenar los contornos por área en orden descendente

contornos = sorted(contornos, key=cv2.contourArea, reverse=True)



# Tomar el contorno de mayor área para el recorte

mayor_contorno = contornos[0]
```

El resultado es el siguiente:

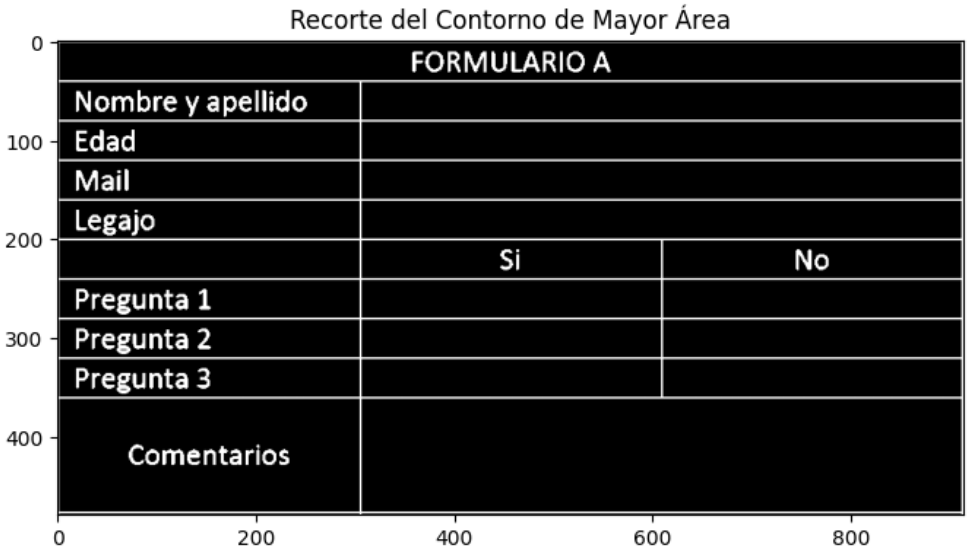
Contorno de Mayor Área en la Imagen

Con el propósito de obtener solamente la porción interior del borde, se utilizó la función "boundingRect" para determinar la posición y las dimensiones del rectángulo que rodea al borde. Posteriormente, se creó una máscara y se procedió a recortar la imagen de acuerdo con el contorno del borde. La implementación fue la siguiente:

```
# Obtener las dimensiones del mayor contorno

x, y, w, h = cv2.boundingRect(mayor_contorno)
```

```
# Crear una máscara en blanco del tamaño del mayor contorno

mascara = np.zeros((h, w), dtype=np.uint8)



# Dibujar el contorno de mayor área en la máscara

cv2.drawContours(mascara, [mayor_contorno], -1, 255,
thickness=cv2.FILLED)



# Recortar el interior del contorno de la imagen original

recorte_contorno_mayor_area = img_umbralizada_invertida[y:y+h, x:x+w]
```

Resultando en la siguiente imagen recortada:



Para obtener la ubicación de las celdas, primero se realizó una detección de bordes utilizando el algoritmo Canny, con umbrales de 50 y 200 para la detección de bordes mínimo y máximo. Los píxeles con intensidades de gradiente por debajo del umbral mínimo se consideran no bordes, mientras que los píxeles con intensidades de gradiente por encima del umbral máximo se consideran bordes fuertes. Los parámetros "None" y "3" son argumentos opcionales que controlan el suavizado gaussiano y el tamaño del kernel. Luego, se realizó una conversión a imagen en color para mostrar las líneas detectadas en rojo y se

utilizó la función "HoughLinesP" para implementar una detección de líneas utilizando la transformada de Hough probabilística. Los parámetros controlan la resolución espacial (1 píxel), resolución angular (np.pi / 180 radianes), umbral de detección (100), "None" indica que no se necesita un array de salida específico, 50 es la longitud mínima de la línea, y 4 es la brecha máxima entre segmentos de línea que se pueden conectar para formar una línea única. El resultado es el siguiente:

Una vez obtenidas las líneas, se calcularon sus intersecciones para cada par de líneas utilizando la regla de Cramer, y se verificó si la intersección calculada se encontraba dentro de los límites de la imagen. La implementación fue la siguiente:

```
intersecciones = []



if linesP is not None:

    for i in range(len(linesP)):

        for j in range(i + 1, len(linesP)):

            line1 = linesP[i][0]

            line2 = linesP[j][0]
```

```
            # Calcular la intersección entre las dos líneas

            x1, y1, x2, y2 = line1

            x3, y3, x4, y4 = line2



            det = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)



            if det != 0:

                x = int(((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) *
(x3 * y4 - y3 * x4)) / det)

                y = int(((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) *
(x3 * y4 - y3 * x4)) / det)



                # Verificar si la intersección está dentro de la imagen

                if 0 <= x < img_umbralizada.shape[1] and 0 <= y <
img_umbralizada.shape[0]:

                    intersecciones.append((x, y))
```

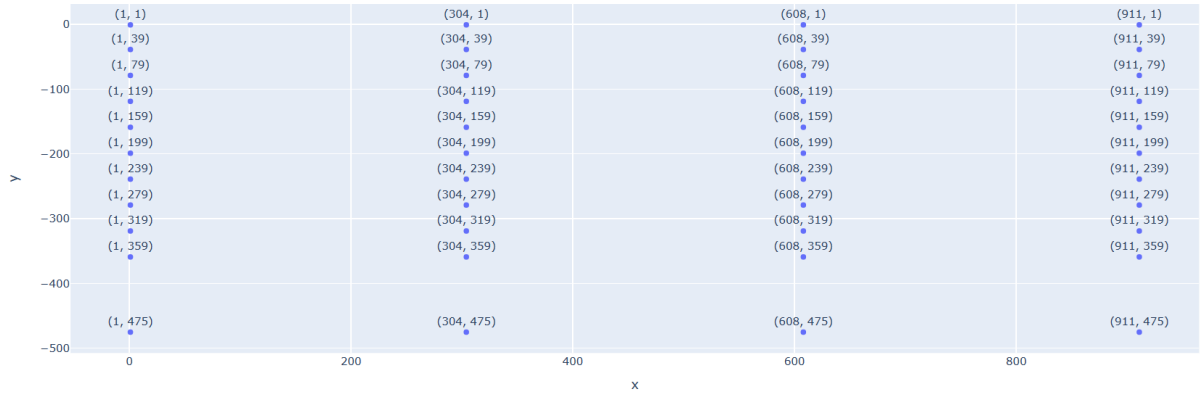
Resultando en las siguientes intersecciones.

Para obtener con precisión las coordenadas de los puntos, se utilizó la biblioteca "Plotly", que se demostró como una herramienta más amigable para esta tarea. Se procedió a representar gráficamente las intersecciones detectadas. En este proceso, se invirtió el eje Y debido a que, en las imágenes, el eje vertical se encuentra orientado hacia abajo. La implementación fue la siguiente:

```
import plotly.express as px

# Invertir los valores de Y

intersecciones_invertidas = [(x, -y) for x, y in intersecciones]



# Extraer las coordenadas x e y de los puntos

x_coords, y_coords = zip(*intersecciones_invertidas)



# Crear una figura de dispersión con Plotly

fig = px.scatter(x=x_coords, y=y_coords, text=[f'({x}, {-y})' for x, y
in intersecciones_invertidas])
```

```
# Configurar etiquetas y título

fig.update_traces(textposition='top center')

fig.update_layout(title='Puntos Fusionados')



# Mostrar la figura

fig.show()
```

El resultado obtenido se presenta a continuación:



Al observar el resultado y el formulario de origen, se creó el siguiente diccionario con cada campo como etiqueta y las coordenadas iniciales y finales en el eje X y en Y, respectivamente:

```
# Coordenadas en la forma x inicial, x final, y inicial, y final

coord_celdas = {'nombre y apellido': (304,911,39,79),

                'edad': (304,911,79,119),

                'mail': (304,911,119,159),

                'legajo': (304,911,159,199),

                'pregunta1 si': (304,608,239,279),

                'pregunta1 no': (608,911,239,279),
```

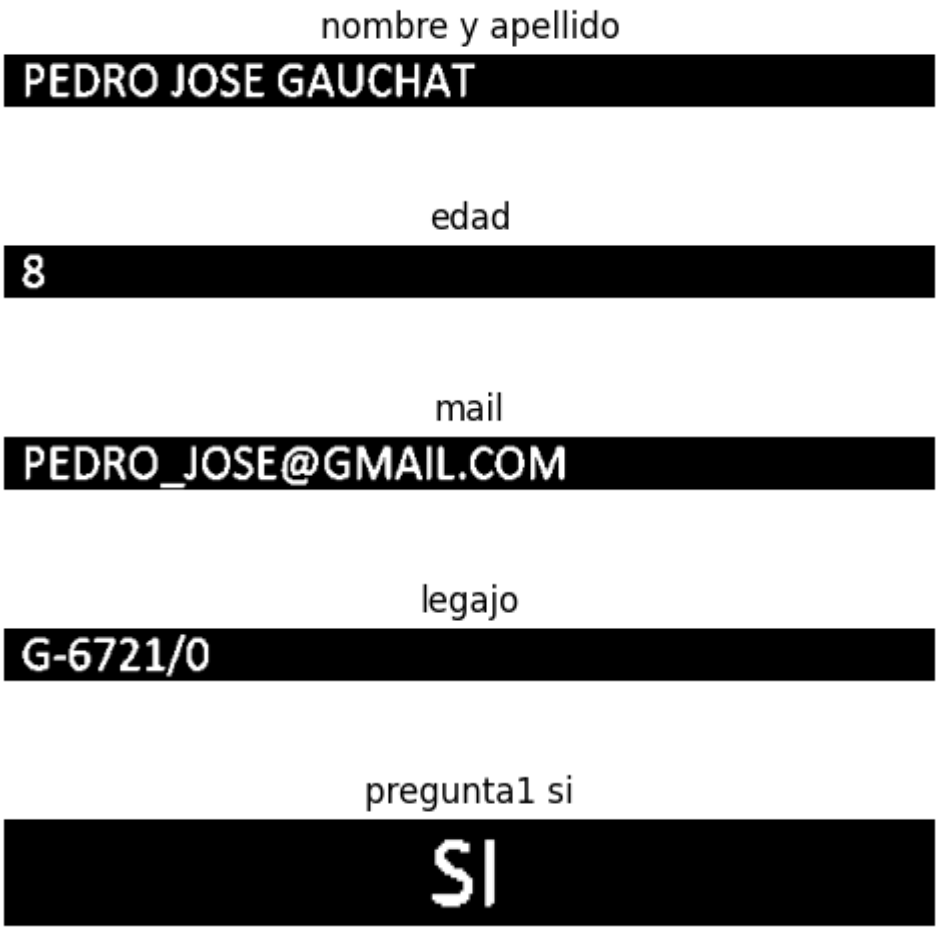
```
                'pregunta2 si': (304,608,279,319),

                'pregunta2 no': (608,911,279,319),

                'pregunta3 si': (304,608,319,359),

                'pregunta3 no': (608,911,319,359),

                'comentarios' : (304,911,359,475)

                }
```

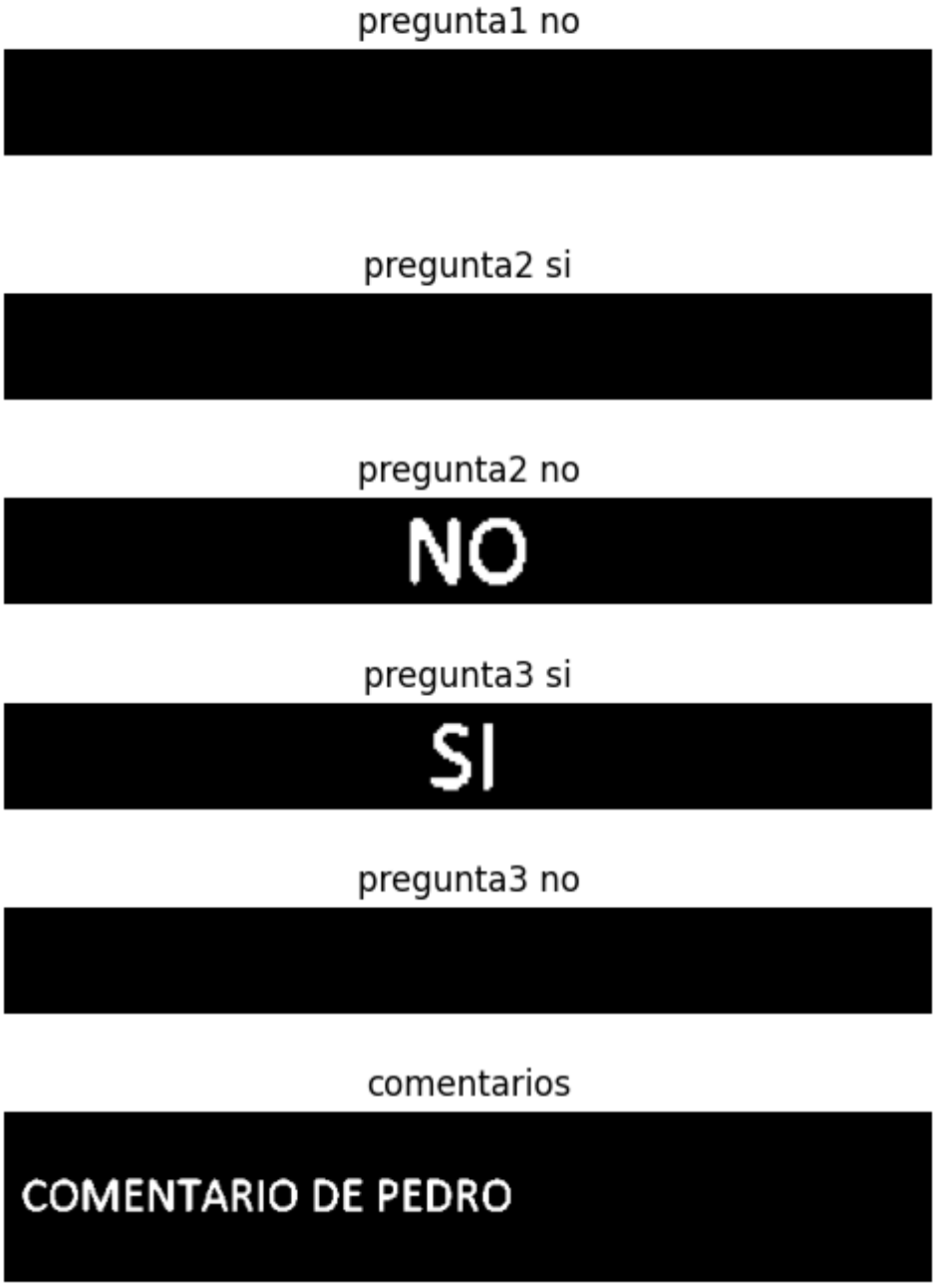
Posteriormente, se utilizó el diccionario para recortar las celdas del formulario, y se realizó un recorte de los bordes superior e izquierdo para eliminar los bordes que quedaron después del recorte. La implementación fue la siguiente:

```
# Recortar y mostrar cada celda

for etiqueta, coordenadas in coord_celdas.items():

    x_inicial, x_final, y_inicial, y_final = coordenadas



    # Recortar la celda de la imagen original

    celda_recortada = recorte_contorno_mayor_area[y_inicial:y_final,
x_inicial:x_final]

# Crear diccionario con celdas

cell_dict_lleno = {}



# Recortar y mostrar cada celda

for etiqueta, coordenadas in coord_celdas.items():

    x_inicial, x_final, y_inicial, y_final = coordenadas
```

```
    # Recortar la celda de la imagen original

    celda_recortada =
recorte_contorno_mayor_area_llena[y_inicial:y_final, x_inicial:x_final]



    # almacenar la celda

    cell_dict_lleno[etiqueta] = celda_recortada

# Recorrer el diccionario y recortar los bordes de las imágenes

for etiqueta, imagen in cell_dict_lleno.items():

    cell_dict_lleno[etiqueta] = imagen[5:-1, 5:-1]
```

El resultado de aplicarlo a la imagen formulario_05.png, por ejemplo, es:





Para continuar, se establecieron los valores de área mínima y máxima permitida para las componentes conexas. Esto se utilizará más adelante para filtrar las componentes que no cumplan con estas restricciones. Se utilizó "connectedComponentsWithStats" para identificar las componentes conexas en la imagen de la celda. Este método devuelve información sobre las componentes, como el número de componentes, las etiquetas de píxeles, estadísticas (como posición y área) y centroides.

Dado que se sabe que las componentes corresponden a palabras en una sola línea, las estadísticas de las componentes se ordenaron en función de su coordenada X izquierda. Esto facilitará el procesamiento de las componentes de izquierda a derecha en la imagen, asegurando que una componente conexa sea el siguiente carácter en el texto.

Se estableció un umbral de proximidad para determinar si dos caracteres están lo suficientemente cerca como para ser considerados parte de la misma palabra.

En un bucle, se recorrieron las estadísticas de las componentes ordenadas, se verificó si el área de la componente cumplía con las restricciones de área definidas anteriormente, y si era así, se incrementó el contador de caracteres ("num_caracteres"). Luego, se verificó si el carácter estaba cerca de la letra de una palabra existente dentro del umbral de proximidad. Si es así, se agrupó con esa palabra; de lo contrario, se inició una nueva palabra.

Finalmente, se creó un diccionario con la cantidad de caracteres y palabras encontradas. La implementación fue la siguiente:

```
# Configura el límite de área mínima y máxima según tus necesidades

    min_area = 10  # Área mínima permitida

    max_area = 500  # Área máxima permitida



    # Aplicar connectedComponentsWithStats para encontrar componentes
conexas

    num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(imagen_grayscale)



    # Ordenar las estadísticas por la coordenada x izquierda

    stats_ordenadas = sorted(stats, key=lambda x: x[cv2.CC_STAT_LEFT])



    # Umbral de proximidad

    umbral_proximidad = 15



    # Contador para el número de componentes conexas dentro del límite
de tamaño

    num_caracteres = 0
```

```
    # Contador para el número de grupos de componentes
    num_palabras = 0

    # Lista para realizar un seguimiento de los grupos y sus componentes
    palabras = []

    # Graficar los centroides de las componentes conexas y contar las válidas
    for i, st in enumerate(stats_ordenadas):

        area = st[cv2.CC_STAT_AREA]

        # Si cumple con las restricciones de área, graficar su centroide y contarlo
        if min_area <= area <= max_area:
            # Coordenadas del centroide
            centroide_x = st[cv2.CC_STAT_LEFT] + st[cv2.CC_STAT_WIDTH] // 2
            centroide_y = st[cv2.CC_STAT_TOP] + st[cv2.CC_STAT_HEIGHT] // 2
            cv2.circle(im_color, (centroide_x, centroide_y), 2, color=(255, 255, 255), thickness=2)
            num_caracteres += 1
```

```
            # Comprobar si este caracter está cerca de la letra de una
palabra existente

            if len(palabras) == 0:

                palabras.append([i])

                num_palabras += 1



            elif abs(centroide_x -
(stats_ordenadas[palabras[num_palabras - 1][-1]][cv2.CC_STAT_LEFT] +

                                    stats_ordenadas[palabras[num_palabras
- 1][-1]][cv2.CC_STAT_WIDTH])) <= umbral_proximidad:



                palabras[num_palabras - 1].append(i)



            else:

                palabras.append([i])

                num_palabras += 1

    return {'caracteres':num_caracteres, 'palabras':num_palabras}
```

El resultado gráfico es el siguiente

Por último, se creó una función que contiene las condiciones a aplicar a cada una de las celdas en cuanto a la cantidad de caracteres y palabras que poseen. La función es la siguiente:

```
def eval_form(cell_dict):

    # Crear diccionario de salida

    output_dict = {}



    # Llenar el diccionario de salida con el conteo de caracteres y
palabras

    for etiqueta, celda in cell_dict.items():

      output_dict[etiqueta] = char_n_word(celda)



    # Condición Nombre y Apellido

    output_dict['nombre y apellido'] = 'OK' if (output_dict['nombre y
apellido']['palabras'] >= 2 and output_dict['nombre y
apellido']['caracteres'] <= 25) else 'MAL'



    # Condición Edad

    output_dict['edad']= 'OK' if (output_dict['edad']['caracteres'] >=
2 and output_dict['edad']['caracteres'] <= 3) else 'MAL'



    # Condición Mail

    output_dict['mail'] = 'OK' if (output_dict['mail']['palabras'] == 1
and output_dict['mail']['caracteres'] <= 25) else 'MAL'



    # Condición Legajo
```

```
    output_dict['legajo'] = 'OK' if
(output_dict['legajo']['caracteres'] == 8) else 'MAL'



    # Condición Comentarios

    output_dict['comentarios'] = 'OK' if
(output_dict['comentarios']['caracteres'] <= 25) else 'MAL'



    # Condición Preguntas

    quest_nums = ['1','2','3']



    for num in quest_nums:

        dict_key = 'pregunta' + num

        dict_key_si = 'pregunta' + num + ' si'

        dict_key_no = 'pregunta' + num + ' no'



        if output_dict[dict_key_si]['caracteres'] > 0 and
output_dict[dict_key_no]['caracteres'] > 0:

            output_dict[dict_key] = 'MAL'

        elif output_dict[dict_key_si]['caracteres'] == 1 or
output_dict[dict_key_no]['caracteres'] == 1:

            output_dict[dict_key] = 'BIEN'

        else:

            output_dict[dict_key] = 'MAL'
```

```
        output_dict.pop(dict_key_si)

        output_dict.pop(dict_key_no)



    return output_dict



eval_form(cell_dict_lleno)
```

El resultado es un diccionario que contiene las condiciones de evaluación para cada celda: `{'nombre y apellido': 'OK', 'edad': 'MAL', 'mail': 'MAL', 'legajo': 'OK', 'comentarios': 'OK', 'pregunta1': 'MAL', 'pregunta2': 'MAL', 'pregunta3': 'MAL'}`

Finalmente, se crearon funciones para organizar y modularizar el código.

2.4 - Implementación detallada del código modularizado

El desarrollo se realizó en Python, utilizando las librerías numpy y opencv-contrib-Python. Se definieron una serie de funciones para llevar a cabo cada etapa del proceso:

- recorte_celdas_form: Esta función segmenta las celdas del formulario y devuelve un diccionario con las celdas recortadas.
- char_n_word: Calcula el número de caracteres y palabras en una celda.
- eval_form: Evalúa si una celda cumple con las restricciones y devuelve un diccionario con el estado de cada campo.
- img_to_validation: Esta función combina las etapas anteriores, toma una imagen del formulario, realiza el recorte y la evaluación de las celdas.

Estructura de la funciones implementadas: Archivo `funciones.py`

```
## Función para recortar formulario en celdas

def recorte_celdas_form(img):



    # Se aplica umbralización sobre la imagen

    umbral = 200
```

```
    img_umbralizada_llena = cv2.threshold(img, umbral, 255,
cv2.THRESH_BINARY)[1]



    # Se invierten los colores (si los bordes están en negro)

    img_umbralizada_llena_invertida = cv2.bitwise_not(img_umbralizada_llena)



    # Se detectan los contornos en la imagen umbralizada invertida

    contornos, _ = cv2.findContours(img_umbralizada_llena_invertida,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)



    # Se ordenan dichos contornos por área en orden descendente

    contornos = sorted(contornos, key=cv2.contourArea, reverse=True)



    # Se toma el contorno de mayor área para el recorte

    mayor_contorno_llena = contornos[0]



    # Se dibuja solo el contorno de mayor área en la imagen original

    img_contorno_mayor_area = cv2.cvtColor(img_umbralizada_llena_invertida,
cv2.COLOR_GRAY2BGR)

    cv2.drawContours(img_contorno_mayor_area, [mayor_contorno_llena], 0, (0,
0, 255), 2)



    # Se obtienen las dimensiones del mayor contorno

    x, y, w, h = cv2.boundingRect(mayor_contorno_llena)
```

```
    # Se crea una máscara en blanco del tamaño del mayor contorno
    mascara = np.zeros((h, w), dtype=np.uint8)

    # Se dibuja el contorno de mayor área en la máscara
    cv2.drawContours(mascara, [mayor_contorno_llena], -1, 255,
thickness=cv2.FILLED)

    # Se recorta el interior del contorno de la imagen original
    recorte_contorno_mayor_area_llena = img_umbralizada_llena_invertida[y:y+h,
x:x+w]

    # Se establecen las coordenadas en la forma x inicial, x final, y inicial,
y final
    coord_celdas = {'Nombre y Apellido': (304,911,39,79),
                    'Edad': (304,911,79,119),
                    'Mail': (304,911,119,159),
                    'Legajo': (304,911,159,199),
                    'pregunta1 si': (304,608,239,279),
                    'pregunta1 no': (608,911,239,279),
                    'pregunta2 si': (304,608,279,319),
                    'pregunta2 no': (608,911,279,319),
                    'pregunta3 si': (304,608,319,359),
                    'pregunta3 no': (608,911,319,359),
```

```
                    'Comentarios' : (304,911,359,475)

                    }



    # Se inicializa el diccionario que contendrá las celdas

    cell_dict_lleno = {}



    # Se recorta y almacena cada celda

    for etiqueta, coordenadas in coord_celdas.items():

        x_inicial, x_final, y_inicial, y_final = coordenadas



        # Se recorta la celda de la imagen original

        celda_recortada = recorte_contorno_mayor_area_llena[y_inicial:y_final,
x_inicial:x_final]



        # Se almacena la celda recortada en el diccionario

        cell_dict_lleno[etiqueta] = celda_recortada



    # Se recorre el diccionario de celdas y recortan los bordes de las
imágenes

    for etiqueta, imagen in cell_dict_lleno.items():

        cell_dict_lleno[etiqueta] = imagen[5:-1, 5:-1]



    return cell_dict_lleno
```

La función recorte_celdas_form tiene como objetivo principal segmentar una imagen de un formulario en diferentes celdas correspondientes a campos específicos. A continuación, se detalla el proceso paso a paso:

- Cargar la imagen: La función recibe una imagen como parámetro (img) y comienza cargándola.
- Umbralización: Se aplica un proceso de umbralización para separar los píxeles de interés del fondo. En este caso, se utiliza un umbral de 200. Esto significa que los píxeles con intensidad mayor a 200 serán considerados como parte del objeto de interés.
- Invertir colores: Si los bordes del formulario están en negro, se invierten los colores para asegurarse de que el contorno del formulario sea blanco y el fondo negro.
- Encontrar contornos: Se utiliza la función findContours de OpenCV para encontrar los contornos presentes en la imagen umbralizada.
- Ordenar contornos por área: Los contornos se ordenan en orden descendente según su área. Esto es importante para identificar el contorno que corresponde al formulario en sí.
- Seleccionar el mayor contorno: Se selecciona el contorno de mayor área, que corresponde al contorno del formulario.
- Dibujar el contorno seleccionado: Se dibuja el contorno seleccionado en una imagen en color para visualizarlo.
- Obtener dimensiones del contorno: Se obtienen las coordenadas y dimensiones del rectángulo que enmarca el contorno del formulario.
- Crear máscara: Se crea una máscara en blanco del tamaño del contorno seleccionado.
- Dibujar el contorno en la máscara: Se dibuja el contorno seleccionado en la máscara.
- Recortar el interior del contorno: Se utiliza la máscara para recortar el interior del contorno de la imagen original.
- Definir coordenadas de las celdas: Se definen las coordenadas de las celdas correspondientes a campos como 'Nombre y Apellido', 'Edad', 'Mail', 'Legajo', etc.
- Recortar y almacenar cada celda: Se recortan las celdas a partir de la imagen del formulario y se almacenan en un diccionario (cell_dict_lleno) donde las etiquetas son los nombres de los campos.
- Recortar bordes de las imágenes: Se realiza un recorte adicional de los bordes de las imágenes almacenadas en el diccionario para que no resulten una dificultad más adelante.
- Finalmente, la función devuelve el diccionario cell_dict_lleno que contiene las celdas correspondientes a cada campo del formulario.

Este proceso permite segmentar la imagen del formulario en celdas individuales para su posterior análisis.

```
## Función para determinar cantidad de caracteres y palabras
```

```python
def char_n_word(imagen_grayscale):

    # Se configura el límite de área mínima y máxima

    min_area = 10  # Área mínima permitida

    max_area = 500  # Área máxima permitida



    # Se aplica connectedComponentsWithStats para encontrar componentes conexas

    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(imagen_grayscale)



    # Se ordenan las estadísticas por la coordenada x izquierda

    stats_ordenadas = sorted(stats, key=lambda x: x[cv2.CC_STAT_LEFT])



    # Se define el umbral de proximidad

    umbral_proximidad = 16



    # Se inicializa el contador para el número de componentes conexas dentro del límite de tamaño

    num_caracteres = 0



    # e inicializa el contador para el número de grupos de componentes

    num_palabras = 0
```

```
    # Se inicializa la lista para realizar un seguimiento de los grupos y sus componentes

    palabras = []



    # Se grafican los centroides de las componentes conexas y se cuentan las válidas

    for i, st in enumerate(stats_ordenadas):



        area = st[cv2.CC_STAT_AREA]



        # Si cumple con las restricciones de área, se grafica su centroide y se aumenta el contador

        if min_area <= area <= max_area:

            # Coordenadas del centroide

            centroide_x = st[cv2.CC_STAT_LEFT] + st[cv2.CC_STAT_WIDTH] // 2

            num_caracteres += 1



            # Se comprueba si este caracter está cerca de la letra de una palabra existente

            if len(palabras) == 0:

                palabras.append([i])

                num_palabras += 1



            elif abs(centroide_x - (stats_ordenadas[palabras[num_palabras - 1][-1]][cv2.CC_STAT_LEFT] +
```

```
                                stats_ordenadas[palabras[num_palabras -
1][-1]][cv2.CC_STAT_WIDTH])) <= umbral_proximidad:



                palabras[num_palabras - 1].append(i)



            else:

                palabras.append([i])

                num_palabras += 1



    return {'caracteres':num_caracteres, 'palabras':num_palabras}
```

La función char_n_word tiene como objetivo contar el número de caracteres y palabras en una imagen en escala de grises. A continuación, se detalla el proceso paso a paso:

1. Configuración de áreas mínima y máxima:
    - min_area = 10 indica el área mínima permitida para considerar un componente conectado.
    - max_area = 500 indica el área máxima permitida.
2. Aplicación de connectedComponentsWithStats:
    - Se aplica la función connectedComponentsWithStats de OpenCV para encontrar componentes conectados en la imagen. Esta función retorna el número de etiquetas, las etiquetas mismas, estadísticas y centroides de los componentes conectados.
3. Ordenamiento de estadísticas:
    - Las estadísticas se ordenan en función de la coordenada x izquierda. Esto permite procesar los componentes conectados de izquierda a derecha.
4. Umbral de proximidad:
    - umbral_proximidad = 16 se utiliza para determinar la proximidad entre caracteres de una misma palabra. Si la diferencia en coordenadas x entre el centroide de dos componentes es menor o igual a 16, se considera que ambos pertenecen a la misma palabra.
5. Contadores y listas:
    - num_caracteres se inicializa como 0 y contendrá el número total de caracteres encontrados.
    - num_palabras se inicializa como 0 y contendrá el número total de palabras encontradas.

- palabras es una lista que servirá para rastrear los grupos de componentes conectados que forman palabras.

6. Iteración sobre componentes conectados:
   - Se itera sobre las estadísticas ordenadas.
   - Para cada componente:
     - Se verifica si su área cumple con las restricciones (min_area y max_area). Si es así, se procede a contar el carácter.
     - Se calculan las coordenadas del centroide.
     - Se verifica si este carácter está cerca de la letra de una palabra existente. Si no hay palabras aún, se crea una nueva lista de componentes para esa palabra. Si hay una palabra existente, se verifica la proximidad y se agrega el componente al grupo correspondiente.
7. Retorno del resultado:
   - Se retorna un diccionario que contiene el número total de caracteres ('caracteres') y el número total de palabras ('palabras').

Esta función es esencial para el proceso de validación de formularios, ya que permite analizar cada componente y determinar si cumple con las restricciones de tamaño y proximidad necesarias para considerarse un carácter válido.

```
## Función de evaluación

def eval_form(cell_dict):

    # Se inicializa el diccionario de salida

    output_dict = {}



    # Se llena el diccionario de salida con el conteo de caracteres y palabras

    for etiqueta, celda in cell_dict.items():

        output_dict[etiqueta] = char_n_word(celda)



    # Condición Nombre y Apellido

    output_dict['Nombre y Apellido'] = 'OK' if (output_dict['Nombre y
Apellido']['palabras'] >= 2 and output_dict['Nombre y Apellido']['caracteres']
<= 25) else 'MAL'
```

```
    # Condición Edad

    output_dict['Edad']= 'OK' if (output_dict['Edad']['caracteres'] >= 2 and
output_dict['Edad']['caracteres'] <= 3) else 'MAL'



    # Condición Mail

    output_dict['Mail'] = 'OK' if (output_dict['Mail']['palabras'] == 1 and
output_dict['Mail']['caracteres'] <= 25) else 'MAL'



    # Condición Legajo

    output_dict['Legajo'] = 'OK' if (output_dict['Legajo']['caracteres'] == 8)
else 'MAL'



    # Condición Comentarios

    output_dict['Comentarios'] = 'OK' if
(output_dict['Comentarios']['caracteres'] <= 25) else 'MAL'



    # Condición Preguntas

    quest_nums = ['1','2','3']



    for num in quest_nums:

        dict_key = 'Pregunta ' + num

        dict_key_si = 'pregunta' + num + ' si'

        dict_key_no = 'pregunta' + num + ' no'
```

```
        if output_dict[dict_key_si]['caracteres'] > 0 and
output_dict[dict_key_no]['caracteres'] > 0:

            output_dict[dict_key] = 'MAL'

        elif output_dict[dict_key_si]['caracteres'] == 1 or
output_dict[dict_key_no]['caracteres'] == 1:

            output_dict[dict_key] = 'OK'

        else:

            output_dict[dict_key] = 'MAL'



        output_dict.pop(dict_key_si)

        output_dict.pop(dict_key_no)




    # Se corrige la ubicación de la condición Comentarios en el diccionario

    temp = output_dict['Comentarios']

    output_dict.pop('Comentarios')

    output_dict['Comentarios'] = temp

    return output_dict
```

La función eval_form tiene como objetivo evaluar los campos de un formulario a partir de un diccionario de celdas previamente recortadas. A continuación, se explica paso a paso lo que hace la función:

1. Creación del diccionario de salida:

    - Se crea un diccionario vacío llamado output_dict que se utilizará para almacenar los resultados de la evaluación.
2. Recorrido de celdas:
    - Se itera sobre el diccionario de celdas cell_dict que contiene las imágenes recortadas de los campos del formulario.
3. Conteo de caracteres y palabras:
    - Para cada etiqueta y su celda correspondiente, se llama a la función char_n_word para contar el número de caracteres y palabras en la celda. Estos resultados se almacenan en output_dict.
4. Condiciones específicas por campo:
    - Se aplican condiciones específicas para cada campo del formulario:
      - Nombre y Apellido: Debe tener al menos 2 palabras y no más de 25 caracteres en total.
      - Edad: Debe tener 2 o 3 caracteres.
      - Mail: Debe contener 1 palabra y no más de 25 caracteres.
      - Legajo: Debe tener exactamente 8 caracteres.
      - Comentarios: No debe contener más de 25 caracteres.
5. Condiciones para preguntas:
    - Para cada una de las 3 preguntas, se verifica si tanto la opción "si" como la opción "no" han sido marcadas. Si es el caso, se marca como "MAL". Si solo una de las opciones está marcada, se marca como "OK".
6. Reordenamiento de campos:
    - Se realiza un reordenamiento de los campos para colocar "Comentarios" al final del diccionario de salida.
7. Retorno del resultado:
    - Finalmente, se retorna el diccionario output_dict con los resultados de la evaluación para cada campo del formulario.

Esta función es crucial para la validación de los formularios, ya que aplica las reglas específicas y retorna un diccionario con el estado de cada campo.

```
## Función completa

def img_to_validation(img):

      crop_dict = recorte_celdas_form(img)

      return eval_form(crop_dict)
```

La función img_to_validation es la función principal que integra las funciones anteriores (recorte_celdas_form y eval_form) para validar un formulario a partir de una imagen. Se detalla su funcionamiento:

1. Entrada y recorte de celdas:
    - Recibe como entrada una imagen del formulario (img).

    - Llama a la función recorte_celdas_form con la imagen como argumento.
    - La función recorte_celdas_form recorta y devuelve un diccionario de celdas que representan los campos del formulario.
2. Evaluación de formulario:
    - El diccionario de celdas es pasado como argumento a la función eval_form.
    - La función eval_form retorna un diccionario con el estado de validación para cada campo del formulario.
3. Resultado de validación:
    - El resultado de la evaluación es retornado por la función img_to_validation.

Aplicación de la función: Archivo `ejercicio2.py`

Nota: Con la finalidad de llevar un registro ordenado de las imágenes analizadas, las mismas, junto con sus correspondientes validaciones, se guardan un un archivo pdf. Este archivo se actualiza añadiendo el contenido nuevo cada vez que se evalúa un nuevo formulario. Así, los resultados se encontrarán disponibles en caso de ser requeridos.

Para ello se crearon funciones adicionales auxiliares en el archivo `funciones.py`.
En dicho archivo se puede observar que las funciones auxiliares hacen uso de la función principal detallada anteriormente.

En el archivo `ejercicio2.py`, se importan las funciones necesarias y se hace uso de ellas sobre la imágenes proporcionadas. Los resultados de las validaciones se guardan junto con los formularios en un archivo pdf. Finalmente, se informa que los resultados han sido guardados.

```
from funciones import generar_dicionario_formularios,
guardar_resultados_en_pdf

## Validar los campos del formulario ##

# Ruta de la imagen del formulario

path_imagenes = './files/images_to_analyze/'



# Lista de nombres de archivo de las imágenes

nombres_imagenes =[['formulario_01.png', 'formulario_02.png',
'formulario_03.png'],['formulario_04.png', 'formulario_05.png',
'formulario_vacio.png']]
```

```
# Guardar los resultados en un archivo PDF

resultados, lista_imagenes = generar_dicionario_formularios(path_imagenes,
nombres_imagenes[0])

guardar_resultados_en_pdf(lista_imagenes, resultados,
'./files/results/resultados_validacion_formularios.pdf')



# Actualizar el archivo PDF agregando nuevos resultados

resultados, lista_imagenes = generar_dicionario_formularios(path_imagenes,
nombres_imagenes[1])

guardar_resultados_en_pdf(lista_imagenes, resultados,
'./files/results/resultados_validacion_formularios.pdf')



print('Resultados guardados en
./files/results/resultados_validacion_formularios.pdf')
```

2.5 - Conclusiones técnicas

Se ha logrado detectar formularios que cumplen con todos los requisitos como los que no, evaluando de a una las celdas de los mismos.

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.	

Resultado:
{

'Nombre y Apellido': 'OK',

'Edad': 'OK', 'Mail': 'OK',

'Legajo': 'OK',

'Pregunta 1': 'OK',

'Pregunta 2': 'OK',

'Pregunta 3': 'OK',

'Comentarios': 'OK'

}

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.	

Resultado:
{

'Nombre y Apellido': 'MAL',

'Edad': 'MAL',

'Mail': 'MAL',

'Legajo': 'MAL',

'Pregunta 1': 'MAL',

'Pregunta 2': 'MAL',

'Pregunta 3': 'MAL',

'Comentarios': 'MAL'

}

2.6 - Desafíos encontrados

- Segmentación precisa: Encontrar el umbral adecuado para la umbralización fue crucial para una segmentación precisa de las celdas.
- Optimización de parámetros: Ajustar los parámetros, como el umbral de área mínima y máxima para caracteres, fue un desafío para garantizar que se detectaran todos los caracteres de manera precisa.
- Manejo de palabras: Identificar y agrupar caracteres en palabras fue un proceso delicado, ya que implicaba tener en cuenta la proximidad entre los caracteres.
- Orden y presentación del informe: Organizar y presentar los resultados de una manera clara y comprensible fue fundamental.

3 - Consideraciones finales

Es importante tener en cuenta que las implementaciones mostradas podrían necesitar ajustes o mejoras según los requisitos específicos de diferentes situaciones o tipos de imágenes. Además con un mayor tiempo podrían depurarse los códigos para que sean más prolijos y claros.

4 - Bibliografía

- Apuntes de cátedra de la asignatura Procesamiento de Imágenes I
- Sitios web con la documentación oficial de las librerías utilizadas:
  - OpenCV: https://docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html
  - fpdf: http://www.fpdf.org/en/doc/index.php
  - PyPDF2: https://pypdf2.readthedocs.io/en/3.0.0/
  - numpy: https://numpy.org/doc/stable/user/absolute_beginners.html
  - matplotlib: https://matplotlib.org/stable/users/index.html