FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento de Imágenes I: Trabajo Práctico 1

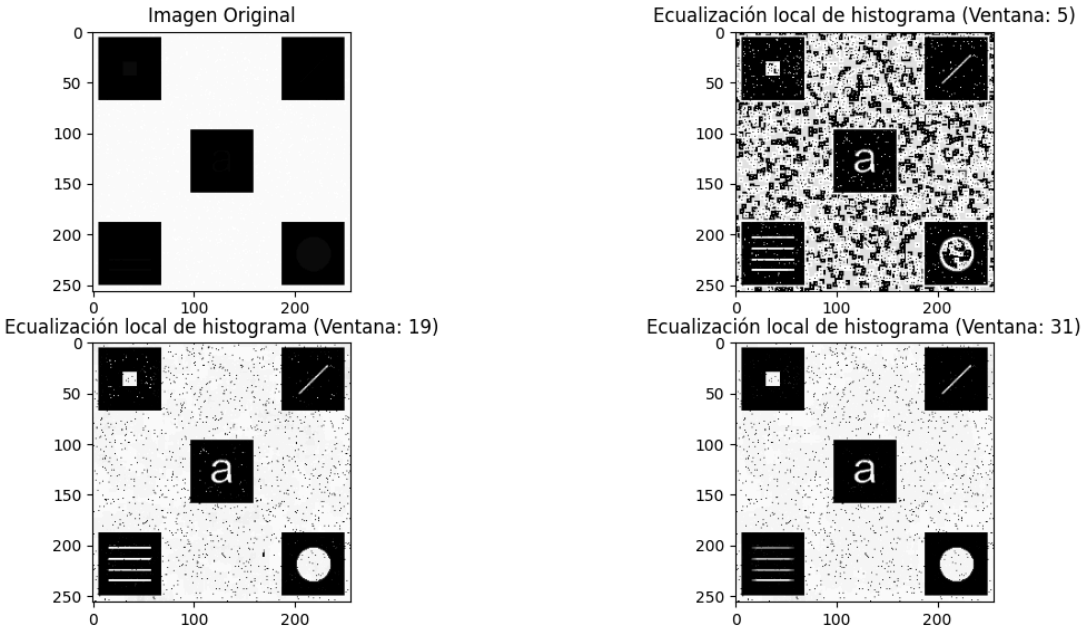
Alumnos:
Facundo Geuna
Pedro Casado
Máximo Alva

Grupo: 5
Año: 2024

Problema 1 - Ecualización local de histograma

Con el propósito de resolver este ejercicio, se utilizaron las siguientes librerías de Python: cv2, numpy, matplotlib.

Para este problema se aplica una ecualizacion local de histograma para detectar detalles escondidos entre los cuadrados negros. Esta técnica sirve para mejorar el contraste en áreas pequeñas de una imagen. Al variar el tamaño de ventana vemos que afecta la claridad con la que vemos los objetos ocultos como también el ruido general que tiene la imagen. En una ventana más pequeña la imagen tiene más ruido pero mayor contraste local, en una ventana más grande la imagen presenta menos ruido pero el contraste local es menor.



Como se observa en la imágenes obtenidas, los detalles escondidos en los cinco cuadrados negros son: un cuadrado pequeño, una barra en diagonal, la letra “a” minúscula, cuatro barras de manera horizontales, y un círculo.

Problema 2 - Corrección de múltiple choice

Con la finalidad de resolver este ejercicio, se utilizaron las siguientes librerías de Python: cv2, numpy, matplotlib.

La resolución de esta problemática fue dividida en cuatro partes, abordando cada uno de los puntos requeridos en el trabajo práctico.

Para el **apartado A**, se confeccionaron cinco funciones diferentes para trabajar de manera organizada y prolija en el código. En primer lugar, se utilizó la función **detectar_linea** para detectar cada una de las líneas verticales y horizontales que nos permiten separar las diferentes preguntas del exámen.

Luego se implementó la función detectar_celda con el objetivo de retornar cada una de las celdas que corresponden con las preguntas del exámen. El diccionario celdas almacena como clave el número de la pregunta y como valor la celda de dicha pregunta.

Siguiendo por este camino, la función detectar_respuesta se encarga de recibir como parámetro de entrada el diccionario celdas y a partir de esto, identificar en la imagen la sección donde se encuentra la respuesta a la pregunta. Para esto, se identifica el renglón donde va la respuesta y se hace un recorte hacía arriba para quedarnos solamente con la sección que nos interesa evaluar. Esto lo conseguimos utilizando la función cv2.connectedComponentsWithStats() que sirve para detectar y analizar componentes conectados en una imagen, lo que nos sirve para obtener las coordenadas del renglón. Luego se retorna el diccionario respuestas con pares 'número de pregunta: 'imagen de la respuesta'

La cuarta función, detectar_letra, a partir de recibir como parámetro el diccionario respuestas, detecta en la imagen si la letra de la respuesta es A, B, C o D. Es importante aclarar que antes de la diferenciación de letras, la función valida que la respuesta solamente tenga una letra. Luego, a partir del número de contornos de cada letra, identifica si se trata de la letra C, B o de A y D. Para estas dos últimas, como tienen el mismo número de contornos, se utilizó el área para diferenciarlas entre sí

Por último, la función evaluar, muestra por pantalla el resultado de cada una de las preguntas del exámen. Para esto se define un diccionario respuestas_correctas que contiene como clave el número de pregunta y como valor la letra correcta. La función evaluar utiliza en su definición tres de las cuatro funciones anteriores: detectar_celda, detectar_respuesta y detectar_letra. Retorna el puntaje del examen utilizando un contador que suma un punto a partir de cada respuesta correcta. Este valor de retorno va a ser utilizado en el apartado C.

Siguiendo con el **apartado B**, donde se pide validar los datos del encabezado de cada examen y mostrar por pantalla el estado de cada campo teniendo en cuenta algunas restricciones, se utilizaron tres funciones. Primero, detectar_name_date_class encuentra los campos 'Name', 'Date', 'Class' y los devuelve recortados de la imagen. Luego, validar_campo encuentra el número de caracteres y de palabras en la imagen y los devuelve. Por último validar_encabezado es la función que llama a las anteriores para realizar la validación completa del encabezado y devolver finalmente el nombre del alumno a evaluar.

En el **apartado C**, se utilizó para mostrar por pantalla los resultados obtenidos de cada uno de los exámenes. Para esto, se confeccionó un diccionario examenes donde la clave es el número del examen y el valor la referencia a la imagen del examen.
Luego, se iniciaron dos diccionarios:

- n_condicion : Un diccionario vacío que se llenará con pares de "número de examen" y su "condición" (Aprobado o Desaprobado).
- n_name: Otro diccionario vacío que guardará las imágenes del campo "Name" de cada examen procesado junto a el número de examen.

Comienza un bucle para todos los exámenes a evaluar.

Primero se realiza la **umbralización** de la imagen:

- Se convierte la imagen en escala de grises en una imagen **binaria**.
- El umbral seleccionado es 190, lo que significa que cualquier píxel con un valor de intensidad mayor o igual a 190 será convertido en 0 (negro) y los píxeles con intensidad menor se convertiran en 1 (blanco).
- Este tipo de umbralización se utiliza para separar las zonas claras (el papel en blanco) de las zonas oscuras (el texto escrito a mano o impreso).

Luego se detectan las columnas y las filas dentro de la imagen umbralizada y se envían a las correspondientes funciones para evaluar el examen.

Se informan los siguientes resultados obtenidos:

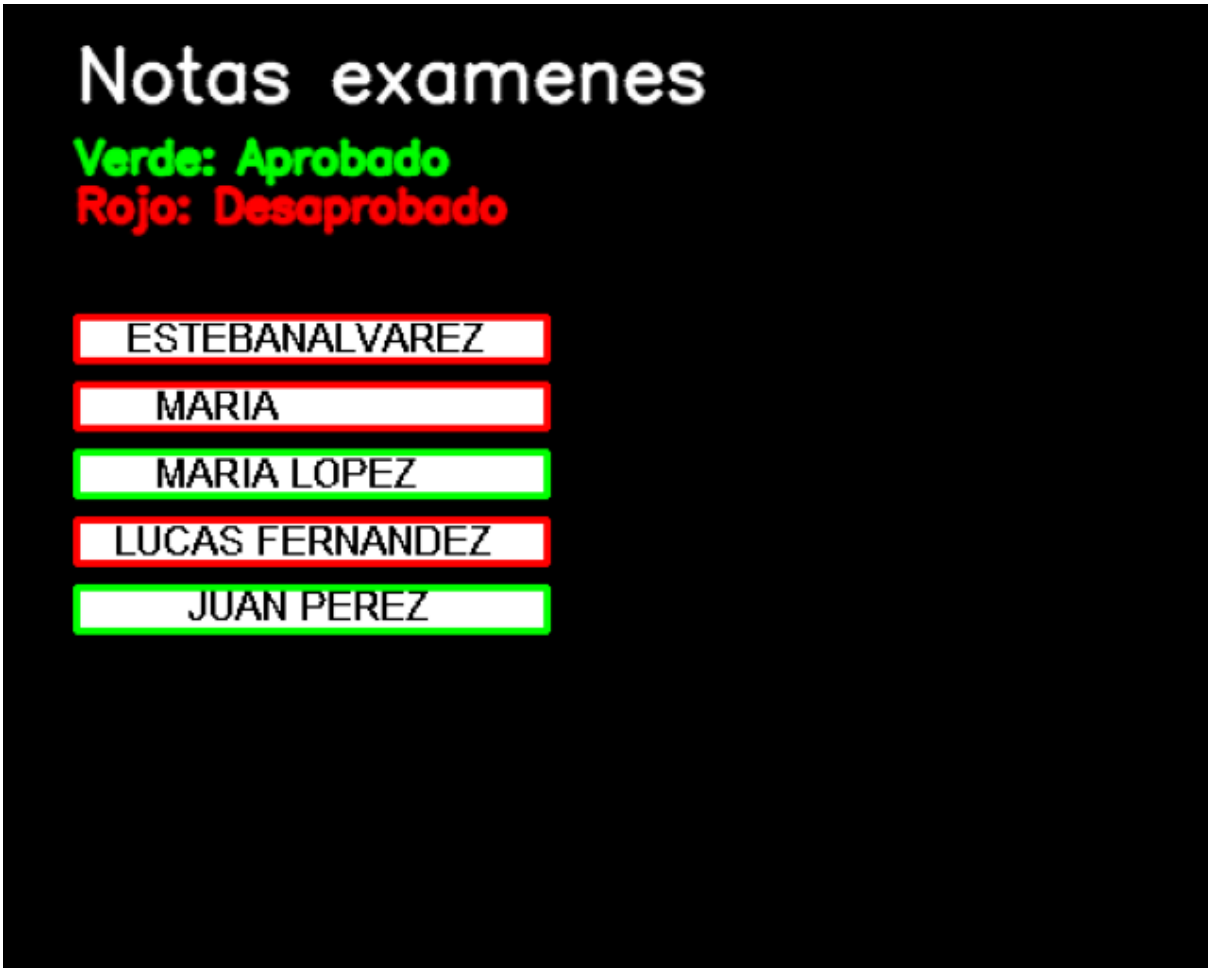
- **Examen 1:**
  Name: MAL
  Date: OK
  Class: OK
  Pregunta 1: MAL
  Pregunta 2: MAL
  Pregunta 3: MAL
  Pregunta 4: MAL
  Pregunta 5: MAL
  Pregunta 6: MAL
  Pregunta 7: MAL
  Pregunta 8: MAL
  Pregunta 9: MAL
  Pregunta 10: MAL
  Calificación: 0
  DESAPROBADO
- **Examen 2:**
  Name: MAL
  Date: OK
  Class: OK
  Pregunta 1: MAL
  Pregunta 2: MAL
  Pregunta 3: MAL
  Pregunta 4: OK
  Pregunta 5: MAL
  Pregunta 6: OK
  Pregunta 7: OK
  Pregunta 8: MAL
  Pregunta 9: MAL
  Pregunta 10: OK
  Calificación: 4
  DESAPROBADO
- **Examen 3:**
  Name: OK

Date: OK
Class: OK
Pregunta 1: OK
Pregunta 2: OK
Pregunta 3: OK
Pregunta 4: OK
Pregunta 5: OK
Pregunta 6: OK
Pregunta 7: OK
Pregunta 8: OK
Pregunta 9: OK
Pregunta 10: OK
Calificación: 10
APROBADO

- **Examen 4:**
  Name: OK
  Date: MAL
  Class: OK
  Pregunta 1: MAL
  Pregunta 2: MAL
  Pregunta 3: MAL
  Pregunta 4: MAL
  Pregunta 5: MAL
  Pregunta 6: MAL
  Pregunta 7: MAL
  Pregunta 8: MAL
  Pregunta 9: MAL
  Pregunta 10: MAL
  Calificación: 0
  DESAPROBADO
- **Examen 5:**
  Name: OK
  Date: OK
  Class: OK
  Pregunta 1: OK
  Pregunta 2: OK
  Pregunta 3: OK
  Pregunta 4: OK
  Pregunta 5: OK
  Pregunta 6: OK
  Pregunta 7: OK
  Pregunta 8: OK
  Pregunta 9: OK
  Pregunta 10: OK
  Calificación: 10
  APROBADO

Por último, para el **apartado D**, se definen los colores y la fuente que se van a utilizar al graficar datos en una nueva imagen. Definimos el título en color blanco, y los subtítulos con los colores verde, para identificar a los alumnos aprobados, y rojo, para aquellos que desaprobaron el examen. Luego para representar el nombre y condición de cada alumno, utilizamos el diccionario n_name ya mencionado anteriormente, de manera que introducimos en la nueva imagen, la sección Name de cada alumno junto con un rectángulo a su alrededor que dependiendo de su condición, este será rojo o verde.
Se adjunta la imagen obtenida.

En este problema aprendimos a obtener información específica a partir de una imagen con formato similar usando funciones y diferentes técnicas de procesamiento de imágenes con las librerías cv2, numpy y matplotlib. Se implementaron funciones específicas para la detección de líneas, celdas y respuestas. La umbralización de las imágenes nos facilitó una división clara entre las zonas de interés: respuestas y campos de datos, facilitando la evaluación automática de los exámenes.



Codigo ejercicio 1

```
import cv2
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Problema 1 - Ecualización local de histograma

# Función 'eq_local_histograma'
def eq_local_histograma(img: np.uint8, size: int) -> np.uint8:
    """
    Aplica la ecualización local del histograma a la imagen.

    Parámetros de entrada:
    img (np.uint8): Imagen a procesar.
    size (int): Tamaño de la ventana de procesamiento.

    Retorna:
    np.uint8: Imagen procesada después de aplicar la ecualización local
del histograma.
    """
    # Padding
    pad_size = size // 2
    pad_img = cv2.copyMakeBorder(img, pad_size, pad_size, pad_size,
pad_size, cv2.BORDER_REPLICATE)

    height, width = img.shape
    # Creamos una copia para no modificar la imagen original
    output_img = img.copy()

    # Iteramos sobre toda la imagen
    for h in range(height):
        for w in range(width):
            # Encuentra ventana de la posición actual y aplica
ecualización de histograma
            ventana = pad_img[h:h + size, w:w + size]
            eq_ventana = cv2.equalizeHist(ventana)
            # Modifica el píxel de la imagen de salida por el central
de la ventana ecualizada
            output_img[h, w] = eq_ventana[pad_size, pad_size]
    return output_img

# Cargar imagen
img = cv2.imread("TP1\Imagen_con_detalles_escondidos.tif",
cv2.IMREAD_GRAYSCALE)
```

```python
# Visualización comparativa de distintos tamaños de ventana
# Imagen Original
ax1 = plt.subplot(221)
plt.imshow(img, cmap='gray')
plt.title('Imagen Original')

# Ventana de tamaño 5
size = 5
eq_img = eq_local_histograma(img, size)
plt.subplot(222, sharex=ax1, sharey=ax1)
plt.imshow(eq_img, cmap='gray')
plt.title(f'Ecualización local de histograma (Ventana: {size})')

# Ventana de tamaño 19
size = 19
eq_img = eq_local_histograma(img, size)
plt.subplot(223, sharex=ax1, sharey=ax1)
plt.imshow(eq_img, cmap='gray')
plt.title(f'Ecualización local de histograma (Ventana: {size})')

# Ventana de tamaño 31
size = 31
eq_img = eq_local_histograma(img, size)
plt.subplot(224, sharex=ax1, sharey=ax1)
plt.imshow(eq_img, cmap='gray')
plt.title(f'Ecualización local de histograma (Ventana: {size})')

plt.show()
```

Codigo ejercicio 2

```python
# Problema 2 - Corrección de multiple choice

# Apartado A
def detectar_linea(pixeles: np.ndarray, umbral: int) -> list:
    """
    Devuelve las líneas detectadas con grosor de un pixel.

    Parámetros de entrada:
    pixeles (np.ndarray): Suma de píxeles en cada columna/fila.
    umbral (int): Valor entero para umbralizar la detección de líneas.
```

```
    Retorna:
    list: Lista con los índices finales de cada línea.
    """
    lineas = pixeles > umbral
    indices_lineas = np.argwhere(lineas).flatten()

    ultimos_indices_lineas = []
    ultimo_indice = indices_lineas[0]
    for i in range(1, len(indices_lineas)):
        # Si la diferencia entre dos índices es mayor que 1, significa
que hemos llegado al final de una línea.
        if indices_lineas[i] - indices_lineas[i - 1] > 1:
            # Guardamos el último índice de la línea actual
            ultimos_indices_lineas.append(ultimo_indice)
        # Actualizamos el último índice
        ultimo_indice = indices_lineas[i]
    # Siempre agregamos el último indice
    ultimos_indices_lineas.append(ultimo_indice)

    return ultimos_indices_lineas


def detectar_celda(img: np.uint8, columnas: list, filas: list) -> dict:
    """
    Devuelve las celdas encontradas entre las filas y columnas.

    Parámetros de entrada:
    img (np.uint8): Imagen del examen.
    columnas (list): Lista con los indices de las columnas.
    filas (list): Lista con los indices de las filas.

    Retorna:
    dict: Diccionario con pares 'número de ejercicio': 'imagen de la
celda'.
    """
    celdas = dict()

    # Recorrer las filas y columnas para extraer las celdas
    n = 1
    for j in range(0, len(columnas) - 1, 2):
        for i in range(len(filas) - 1):
            # Definir el área de la celda como (y1, y2) en filas y (x1,
x2) en columnas
```

```
            y1, y2 = filas[i], filas[i+1]
            x1, x2 = columnas[j], columnas[j+1]
            # Extraer subimagen de la celda
            subimg = img[y1:y2, x1:x2]
            celdas[n] = subimg
            n += 1
            # Mostrar subimagen
            #plt.imshow(subimg, cmap='gray'), plt.title(f'Celda
{n-1}'), plt.show()

    return celdas


def detectar_respuesta(celdas: dict) -> dict:
    """
    Encuentra y devuelve la zona de cada imagen donde está la respuesta
al ejercicio.

    Parámetros de entrada:
    celdas (dict): Diccionario con pares 'número de ejercicio': 'imagen
de la celda'.

    Retorna:
    dict: Diccionario con pares 'número de ejercicio': 'imagen de la
respuesta'.
    """
    respuestas = dict()
    for n, celda in celdas.items():
        num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(celda, 8, cv2.CV_32S)
        for st in stats:
            if st[2] > 50 and st[2] < 150:
                respuestas[n] = celda[st[1]-13:st[1]-2,
st[0]:st[0]+st[2]]

    #for respuesta in respuestas.values():
        #plt.imshow(respuesta, cmap='gray'), plt.show()

    return respuestas


def detectar_letra(respuestas: dict) -> dict:
    """
```

```
    A partir de una imágen detecta si se trata de una A, B, C o D.
Cualquier otro hallazgo es tomado como error.

    Parámetros de entrada:
    respuestas (dict): Diccionario con pares 'número de ejercicio':
'imagen de la respuesta'.

    Retorna:
    dict: Diccionario con pares 'número de ejercicio': 'letra'.
    """
    letras = dict()

    for n, respuesta in respuestas.items():
        num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(respuesta, 8, cv2.CV_32S)
        letra = 'MAL'
        # Valido que solo haya una letra
        if num_labels == 2:
            contornos, _ = cv2.findContours(respuesta, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
            num_contornos = len(contornos)
            if num_contornos == 1:
                letra = "C"
            elif num_contornos == 3:
                letra = "B"
            elif num_contornos == 2: #puede ser A o D, analizamos el
area
                area = cv2.contourArea(contornos[0])
                #print(area)
                if area == 23.0:
                    letra = "A"
                else:
                    letra = "D"
        letras[n] = letra

    return letras


def evaluar(img: np.uint8, columnas: list, filas: list) -> int:
    """
    Muestra en pantalla los resultados de las respuestas del examen
    que reciba como parámetro de entrada y retorna la cantidad de
respuestas correctas.
```

```
    Parámetros de entrada:
    img (np.uint8): Exámen a evaluar.
    columnas (list): Lista con los indices de las columnas.
    filas (list): Lista con los indices de las filas.

    Retorna:
    int: Cantidad de respuestas correctas
    """
    # Diccionario con pares "pregunta": "respuesta".
    respuestas_correctas = {1: 'C', 2: 'B', 3: 'A', 4: 'D', 5: 'B', 6:
'B', 7: 'A', 8: 'B', 9: 'D', 10: 'D'}

    celdas = detectar_celda(img_umbral, columnas, filas)
    respuestas = detectar_respuesta(celdas)
    letras = detectar_letra(respuestas)

    # Calculamos el puntaje y mostramos por pantalla las correciones de
cada pregunta
    puntaje = 0
    for n, letra in letras.items():
        if letra == respuestas_correctas[n]:
            puntaje += 1
            print(f"Pregunta {n}: OK")
        else:
            print(f"Pregunta {n}: MAL")

    return puntaje


# Apartado B
def detectar_name_date_class(img: np.uint8, filas: list) -> list:
    """
    Encuentra los campos 'Name', 'Date', 'Class' y los devuelve
recortados de la imagen.

    Parámetros de entrada:
    img (np.uint8): Imagen del encabezado.
    filas (list): Lista con los indices de las filas.

    Retorna:
    list: lista con las imágenes del encabezado y los campos 'Name',
'Date', 'Class'.
```

```
    """
    # Primero encontramos el encabezado completo
    y1, y2 = 0, filas[0] - 2
    x1, x2 = 0, img.shape[1]
    encabezado = img[y1:y2, x1:x2]
    #plt.imshow(encabezado, cmap='gray'), plt.show()
    campos = list()
    num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(encabezado, 8, cv2.CV_32S)
    for st in stats:
        if st[2] > 50 and st[2] < 200:
            campos.append(encabezado[st[1]-20:st[1]-2,
st[0]:st[0]+st[2]])
    #for campo in campos:
        #plt.imshow(campo, cmap='gray'), plt.show()

    return campos

def validar_campo(img: np.uint8) -> tuple[int, int]:
    """
    Encuentra el número de caracteres y de palabras en la imagen y los
devuelve.

    Parámetros de entrada:
    img (np.uint8): Imagen del campo a analizar.

    Retorna:
    tuple[int, int]: devuelve una tupla con el número de palabras y el
número de caracteres del campo.
    """
    num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(img, 8, cv2.CV_32S)
    n_caracteres = 0
    n_palabras = 0
    x_ant = None
    # Recorrer todas las componentes conectadas, ignorando el fondo
    for i in range(1, num_labels):
        x, y, width, height, area = stats[i]
        n_caracteres += 1
        # Determinar si es parte de una nueva palabra (basado en la
distancia entre letras)
        if x_ant is not None:
            distancia = x - x_ant
```

```
            if distancia > 15:
                n_palabras += 1
        # Actualizar la posición de la última letra
        x_ant = x
    # Contar la última palabra
    if n_caracteres > 0:
        n_palabras += 1

    return n_palabras, n_caracteres

def validar_encabezado(img: np.uint8, filas: list) -> np.uint8:
    """
    Valida los datos del encabezado (Name, Date, Class)
    del examen que recibe como parametro de entrada.

    Parámetros de entrada:
    img (np.uint8): Imagen del encabezado a analizar.

    Retorna: La imágen del campo 'Name'.
    """
    campos = detectar_name_date_class(img, filas)

    # Validación 'Name'
    n_palabras, n_caracteres = validar_campo(campos[0])
    if n_palabras >= 2 and n_caracteres <= 25:
        print("Name: OK")
    else:
        print("Name: MAL")

    # Validación 'Date'
    n_palabras, n_caracteres = validar_campo(campos[1])
    if n_palabras == 1 and n_caracteres == 8:
        print("Date: OK")
    else:
        print("Date: MAL")

    # Validación 'Class'
    _, n_caracteres = validar_campo(campos[2])
    if n_caracteres != 1:
        print("Class: MAL")
    else:
        print("Class: OK")
```

```
    _, name = cv2.threshold(campos[0], 0, 255, cv2.THRESH_BINARY_INV)
    return name


# Apartado C
examenes = {1: 'TP1/examen_1.png',
            2: 'TP1/examen_2.png',
            3: 'TP1/examen_3.png',
            4: 'TP1/examen_4.png',
            5: 'TP1/examen_5.png'}

n_condicion = dict() # diccionario con pares 'Número de examen':
'Condición'
n_name = dict() # diccionario con pares 'Número de examen': 'Imagen dl
campo Name'

for n, examen in examenes.items():
    img = cv2.imread(examen, cv2.IMREAD_GRAYSCALE)
    # Umbralizamos la imagen. Estos umbrales se eligieron por prueba.
    _, img_umbral = cv2.threshold(img, 190, 1, cv2.THRESH_BINARY_INV)
    #plt.imshow(img_umbral, cmap='gray'), plt.show()

    # Sumamos los píxeles de cada columna y fila.
    pixeles_columnas = np.sum(img_umbral, 0)
    pixeles_filas = np.sum(img_umbral, 1)
    # Detectamos las columnas y filas
    columnas = detectar_linea(pixeles_columnas, 600)
    filas = detectar_linea(pixeles_filas, 450)

    print(f"Examen {n}: ")
    n_name[n] = validar_encabezado(img_umbral, filas)
    nota = evaluar(img_umbral, columnas, filas)

    print(f"Calificación: {nota}")
    if nota >= 6:
        condicion = "APROBADO"
        print(f"{condicion}")
    else:
        condicion = "DESAPROBADO"
        print(f"{condicion}")
    n_condicion[n] = condicion
```

```
# Apartado D

img_resultados = np.ones((400, 500,3), dtype = np.uint8)
color_aprobado = (0, 255, 0) # Verde
color_desaprobado = (0, 0, 255) # Rojo
color_titulo = (255, 255, 255) # Blanco
fuente = cv2.FONT_HERSHEY_SIMPLEX
y = 40
# Título
cv2.putText(img_resultados, "Notas examenes", (30, y), fuente, 1,
color_titulo, 2, cv2.LINE_AA)
# Subtítulos para indicar qué representa cada color
y += 30
cv2.putText(img_resultados, "Verde: Aprobado", (30, y), fuente, 0.6,
color_aprobado, 2, cv2.LINE_AA)
y += 20
cv2.putText(img_resultados, "Rojo: Desaprobado", (30, y), fuente, 0.6,
color_desaprobado, 2, cv2.LINE_AA)

y += 40
for (n, condicion), (name) in zip(n_condicion.items(),
n_name.values()):
    if condicion == "APROBADO":
        color = color_aprobado
    else:
        color = color_desaprobado
    # Insertar la imagen en la imagen de resultados
    name = cv2.cvtColor(name, cv2.COLOR_BGRA2BGR)
    img_resultados[y:y + name.shape[0], 30:30 + name.shape[1]] = name

    # Dibujar un recuadro alrededor de la imagen con el color
correspondiente
    cv2.rectangle(img_resultados, (30, y), (30 + name.shape[1], y +
name.shape[0]), color, 2)

    # Aumentar la posición y para la siguiente entrada
    y += name.shape[0] + 10

# Visualizar imagen
plt.imshow(cv2.cvtColor(img_resultados, cv2.COLOR_BGR2RGB)), plt.show()
```