Procesamiento de Imágenes 1

Trabajo Práctico N°1- Informe

Rodríguez Griño, Tomás: R-4643/4

Rosito, Valentín: R-4662/1

Slepoy, David: S-4782/7

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Tecnicatura Universitaria en Inteligencia Artificial.

Problema 1

Para poder resolver el punto 1 del trabajo práctico debíamos realizar una función que reciba una imagen y dos valores valores que representan la dimensión del kernel (M y N).

Una vez tomados estos parámetros la función debe poder a través de la Ecualización poder encontrar detalles. La función expande los bordes de la imagen con **copyMakerBorder**, tomando como parámetros ***M//2*** (No puede tomar valores que no sean números naturales) para agregar en los bordes superior e inferior y ***N//2*** en los bordes izquierdo y derecho. Esto se realiza con el fin de prevenir que cuando el kernel pase por los píxeles de los bordes, no salga de los límites de la imagen.

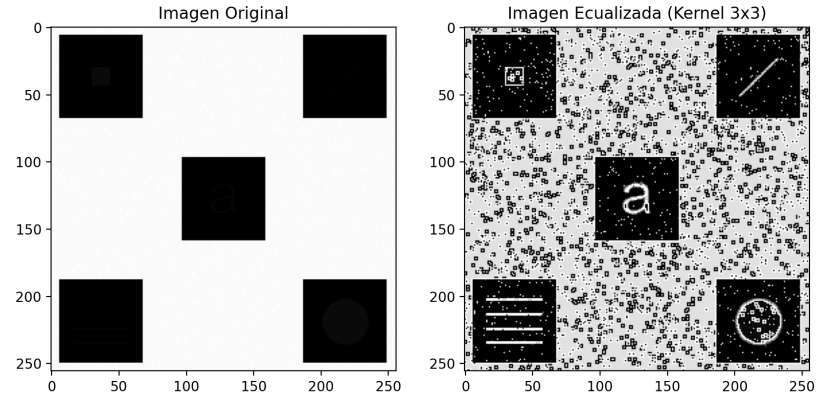
```python
def local_histogram(img, M, N):
    # Extendemos los bordes de la imagen
    img_expand = cv2.copyMakeBorder(img, M//2, M//2, N//2, N//2, cv2.BORDER_REPLICATE)

    # Imagen de salida
    img_equalized = np.zeros_like(img)

    # Deslizamos el kernel por la imagen
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            # Definimos el kernel en (i, j)
            window = img_expand[i:i+M, j:j+N]
            # Ecualizacion del histograma local
            img_equalized[i, j] = cv2.equalizeHist(window)[M//2, N//2]

    return img_equalized

img = cv2.imread("TP1/Imagen_con_detalles_escondidos.tif", cv2.IMREAD_GRAYSCALE)
M, N = 15, 15  # Tamaño del kernel

# Aplicamos la ecualización local del histograma
img_equalized = local_histogram(img, M, N)
```

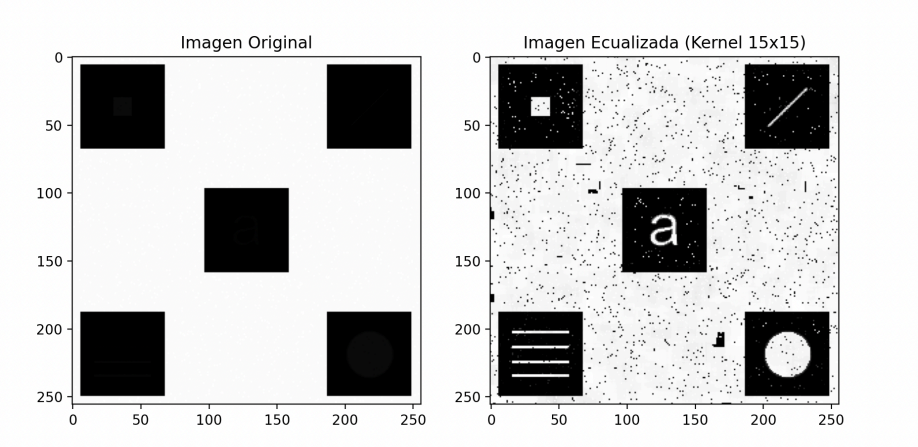
Dentro de cada ventana, se calcula el histograma de los niveles de intensidad de los píxeles y se aplica una transformación para ecualizar esta región localmente. Esto asegura que los detalles que están se puedan ver.

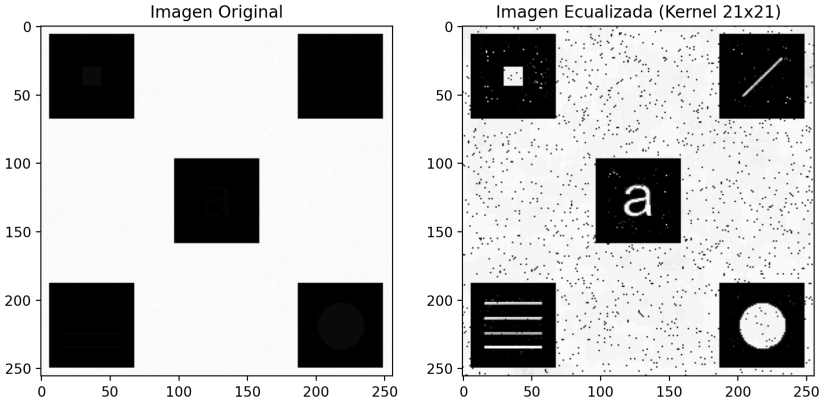
Se usó inicialmente un tamaño de ventana de 3×3. Pudimos observar que con es estos valores para M y N eran muy bajos y modificaban valores fuera de los que pedía el enunciado y no resaltaba con la claridad que pretendíamos las figuras ocultas. Luego fuimos aumentando los

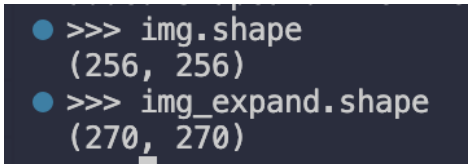
valores hasta llegar a un M x N de 15 x 15 que resalta con claridad los detalles en las zonas más oscuras.

Ejemplos









Este es el resultado de expandir la imagen con una M y N asumiendo el valor 15 (Aumentando 14 pixeles a lo largo y alto por estar redondeada la división de M y N)

En la siguiente página podemos visualizar los **distintos tamaños de kernel aplicados** y que resultado obtuvimos de ellos. Después de analizar las imágenes, **decidimos emplear un kernel 15x15**, ya que nos daba el mejor resultado, sin comprometer detalles de la imagen original.

Conclusiones

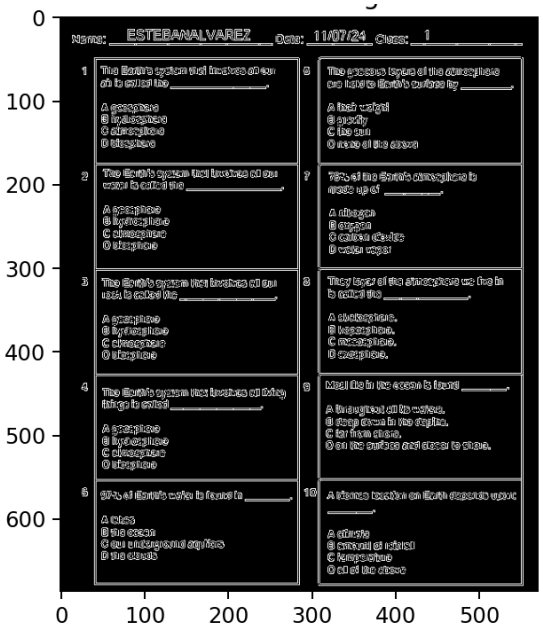
La técnica de ecualización local del histograma demostró ser muy útil para resaltar detalles en zonas con niveles de intensidad similares. A diferencia de la ecualización global, que puede perder detalles locales, este enfoque permite mejorar el contraste en regiones específicas de la imagen. El tamaño de la ventana M×NM \times NM×N es un parámetro crítico y debe ser ajustado en función de las características de la imagen a procesar.

A modo de recomendación, se sugiere experimentar con diferentes tamaños de ventana y realizar un análisis visual de los resultados obtenidos para determinar el tamaño óptimo.

Problema 2

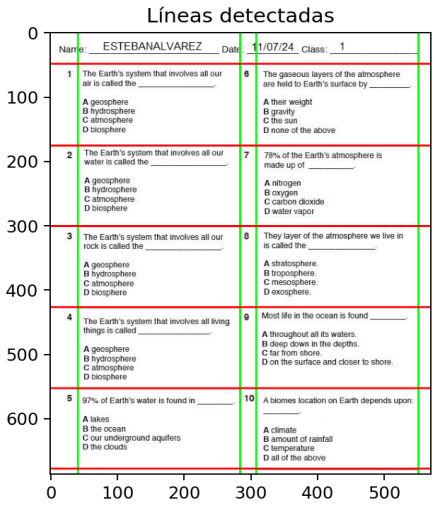
Inciso a)

Comenzamos la resolución de este problema con dudas sobre como detectar las líneas de cada bloque, debatiendo varias ideas, probamos con Canny inicialmente, pero consultando esto, claramente estábamos buscando “matar una mosca con una bomba”

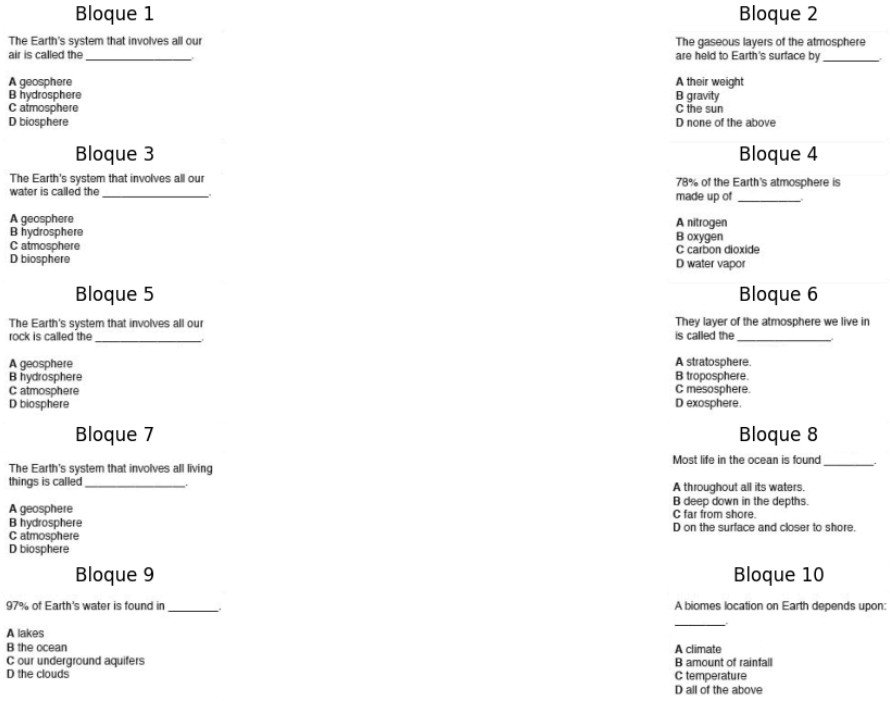


Entonces tomamos la decisión de realizar un umbralado binario. Para luego, sobre la imagen binarizada definir los umbrales de las líneas verticales y horizontales, con el fin de detectar cada bloque de preguntas.
A pesar de que había detectado de manera muy acertada las líneas, encontrábamos un problema, ya que detectaba los espacios entre los bloques (Con números 6, 7, 8, 9, 10). Para solucionar esto, en el ciclo for que creamos con las **variables *i* y *j*** cuando j=1

```python
def detectar_lineas_horizontales(imagen, umbral=int):
    _, img_th = cv2.threshold(imagen, 128, 255, cv2.THRESH_BINARY_INV)
    # Convertimos la imagen binaria a 1 y 0 (Para facilitar la sumas)
    img_th_ones = img_th // 255
    img_fils = np.sum(img_th_ones, axis=1)
    th_fil = np.max(img_fils) * umbral
    imh_fils_th = img_fils > th_fil

    lineas_h = []
    en_linea = False
    for i, val in enumerate(imh_fils_th):
        if val and not en_linea:
            inicio_fil = i
            en_linea = True
        elif not val and en_linea:
            fin_fil = i
            lineas_h.append((inicio_fil, fin_fil))
            en_linea = False
    return lineas_h
```

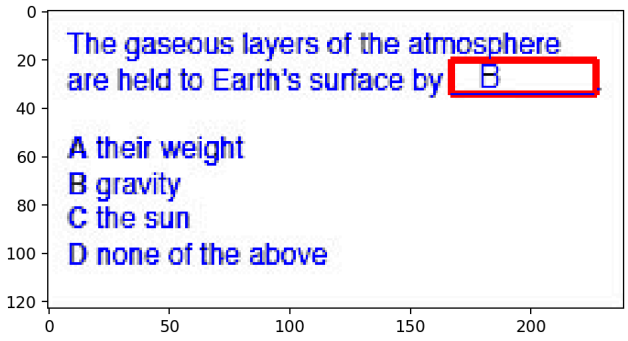


(En el rango del largo de la cantidad de líneas verticales) pusimos un continue con el fin de que no tome esa línea para comenzar el nuevo bloque y tome las coordenadas de j=2 que son las que se deben usar.



Esta función es la encargada de dividir los bloques con las líneas verticales y horizontales que le pasaron y la imagen en la cual se divide.
Luego **dentro de cada bloque** tuvimos que **detectar la línea de respuesta** y generar un box sobre ella, para posteriormente evaluar su contenido.

```python
def division_bloques(lineas_v, lineas_h,img):
    blocks = []
    for i in range(len(lineas_h) - 1):
        for j in range(len(lineas_v) - 1):
            if j == 1:  # Esto excluye la segunda columna (los números del medio)
                continue
            x1 = lineas_v[j][1] + 1
            x2 = lineas_v[j+1][0]- 1
            y1 = lineas_h[i][1] + 1
            y2 = lineas_h[i+1][0]-1
            # Recortamos el bloque
            block = img[y1:y2, x1:x2]
            blocks.append(block)
    blocks = blocks[2:]
    fig, axs = plt.subplots(len(blocks)//2, 2, figsize=(10, 10))
    axs = axs.flatten()
    for i, block in enumerate(blocks):
        axs[i].imshow(block, cmap='gray')
        axs[i].set_title(f'Bloque {i+1}')
        axs[i].axis('off'), axs[i].set_xticks([]), axs[i].set_yticks([])  # Oculta los ejes
    plt.tight_layout()
    plt.show()
    return blocks
```




```python
def detectar_linea_pregunta(imagen):
    ##Hacemos esto ya que la imagen binarizada no encuentra la linea con los contornos
    imagen_entrada = cv2.cvtColor(imagen.copy(), cv2.COLOR_GRAY2BGR)
    _, img_binaria = cv2.threshold(imagen, 128, 255, cv2.THRESH_BINARY_INV)
    contornos, _ = cv2.findContours(img_binaria, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    ancho = 0
    linea = (0, 0, 0, 0)

    for contorno in contornos:
        # Encontramos la linea
        x, y, w, h = cv2.boundingRect(contorno)
        if ancho < w:
            ancho = w
            linea = (x, y, w, h-15)
    x, y, w, h = linea
    cv2.rectangle(imagen_entrada, (x, y), (x + w, y + h), (0, 0, 255), 2)

    # Dibujar los contornos en la imagen
    # cv2.drawContours(imagen_entrada, contornos, -1, (255, 0, 0), 1)
    # plt.imshow(cv2.cvtColor(imagen_entrada, cv2.COLOR_BGR2RGB))
    # plt.show()
    return linea
```

En estas líneas de código, extraemos las coordenadas de las líneas de las preguntas con x,t,w,h de la línea y le restamos 15 a la altura para que nos dé el recuadro completo de cada pregunta cuando la función sea llamada.

.

```python
def detectar_respuesta(imagen, rectangulo):
    blocardo = imagen.copy()
    x, y, w, h = rectangulo
    respuesta = blocardo[y+h:y, x:x+w]
    _, respuesta = cv2.threshold(respuesta, 128, 255, cv2.THRESH_BINARY)

    conectados = 8
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(respuesta, conectados, cv2.CV_32S)

    # Invertimos para que los contornos de los huecos sean detectados
    _, bin_respuesta = cv2.threshold(respuesta, 128, 255, cv2.THRESH_BINARY_INV)

    contornos, hierarchy = cv2.findContours(bin_respuesta, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

    num_hijos = 0
    letra_detectada = ""
    area_hijo = 0
    respuestas_detectadas = []
    # Verificar la jerarquía para contar los hijos del contorno externo (índice 0)
    if hierarchy is not None:
        hijo_actual = hierarchy[0][0][2]  # Obtener el primer hijo del contorno 0
        while hijo_actual != -1:
            num_hijos += 1
            # Calcular el área del contorno del hijo
            area_hijo = cv2.contourArea(contornos[hijo_actual])
            hijo_actual = hierarchy[0][hijo_actual][0]  # Ir al siguiente hermano
```

```python
        if len(contornos) > 3:
            letra_detectada = ""
        elif num_hijos == 0:
            letra_detectada = "C"  # No tiene hijos
        elif num_hijos == 1:
            # Distinguir entre A y D en función del área del hijo
            if area_hijo > 35:  # Umbral
                letra_detectada = "D"
            else:
                letra_detectada = "A"
        elif num_hijos == 2:
            letra_detectada = "B"  # Tiene dos hijos (dos huecos, como la letra B)
    rta_examen.append(letra_detectada)
    # return letra_detectada
```

Detección de letras mediante jerarquía y análisis de contornos:

Para identificar correctamente las letras, utilizamos una función que nos permite analizar los contornos de cada carácter. Implementamos el uso de jerarquía para determinar la cantidad de "hijos" que tiene cada contorno, lo que nos ayuda a identificar la letra correspondiente. Establecimos las siguientes reglas:

- Si un contorno no tiene hijos, lo clasificamos como la letra **C**.
- Si el contorno tiene dos hijos, lo identificamos como la letra **B**.
- Si el contorno tiene un solo hijo, podría ser tanto la letra **A** como la **D**. Para diferenciarlas, utilizamos el área del contorno, ya que el área de la letra **D** es significativamente mayor que la de la **A**.

Por lo tanto, si el área del contorno es mayor a 35 unidades, se clasifica como la letra **D**; de lo contrario, se clasifica como la letra **A**.

Inciso b)

```python
def detectar_caracteres_encabezado(imagen_encabezado):
    imagen_encabezado = cv2.adaptiveThreshold(imagen_encabezado, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)
    contours, hierarchy = cv2.findContours(imagen_encabezado, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    umbral_area = 5  # Definimos el umbral del área que necesitamos
    contornos_filtrados = [cnt for cnt in contours if cv2.contourArea(cnt) > umbral_area]
    umbral_distancia = 15
    espacios = 0

    # Calculamos la distancia entre los contornos filtrados
    for i in range(-1, len(contornos_filtrados) - 1):
        x_actual, _, w_actual, _ = cv2.boundingRect(contornos_filtrados[i])
        x_siguiente, _, w_siguiente, _ = cv2.boundingRect(contornos_filtrados[i + 1])

        # Calculamos la distancia entre los componentes de esta forma ya que toma los contornos del ultimo al primero
        distancia = x_actual - x_siguiente

        # Si la distancia, consideramos que hay un espacio entre palabras
        if distancia >= umbral_distancia:
            espacios += 1
    salida = {"Caracteres": len(contornos_filtrados), "Espacios": espacios, "Palabras": espacios + 1}
    return salida
```

Función `detectar_caracteres_encabezado`:

La función `detectar_caracteres_encabezado` se encarga de identificar los contornos de las letras en el encabezado. Para ello, aplicamos un **umbral de área mínima de 5**, que consideramos adecuado para detectar correctamente las letras de los encabezados.

Una vez identificados los contornos, calculamos las distancias entre ellos y **aplicamos un umbral de 15 para distinguir los espacios entre contornos** (Para poder obtener la cantidad de palabras). Esto nos permite detectar si los contornos pertenecen a palabras diferentes, como en el caso de separar los nombres en el encabezado.

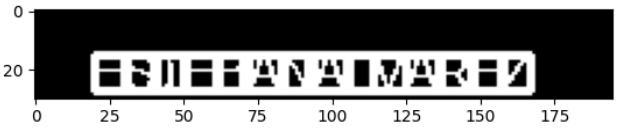
En las siguientes imágenes podemos observar la imagen original de cada campodel encabezado (Nombre, Fecha y Clase) y esta misma luego de haberle aplicado la función `detectar_caracteres_encabezado`.

En la cual se pueden observar dos imágenes por cada campo. La **1)** siendo la imagen original y la **2)** la imagen con los contornos que superen el umbral de área (El cual es igual a 5)

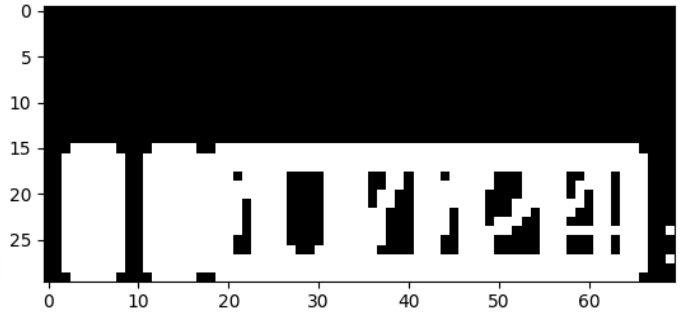
1- Imagen original - Nombre
2- Imagen con los contornos que pasan el umbral - Nombre

1)

2)



1 - Imagen original - Fecha
2 - Imagen con los contornos que pasan el umbral - Fecha



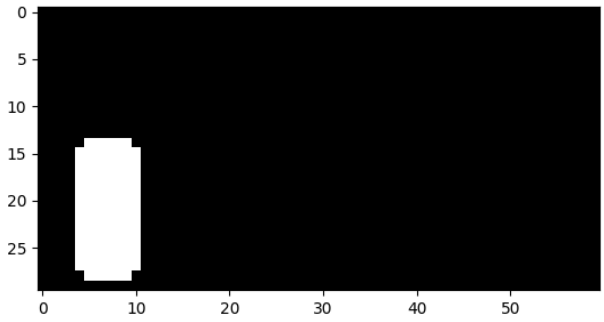
1)

2)

1- Imagen original - Clase
2- Imagen con los contornos que pasan el umbral - Clase

1)

2)



```
nombres = []
examenes = ['examen_1.png','examen_2.png','examen_3.png','examen_4.png', 'examen_5.png']
for examen in examenes:
    rta_examen = []
    img = cv2.imread(examen, 2)
    # Detectar líneas verticales y horizontales
    lineas_v = detectar_lineas_verticales(img, 0.6)
    lineas_h = detectar_lineas_horizontales(img, 0.5)
    blocks = division_bloques(lineas_v, lineas_h, img)

    # Corto cada campo
    nombre = img[0:30, 60:255]
    fecha = img[0:30, 300:370]
    clase = img[0:30, 430:490]


    nombres.append(nombre)

    d_nombre = detectar_caracteres_encabezado(nombre)
    d_fecha = detectar_caracteres_encabezado(fecha)
    d_class = detectar_caracteres_encabezado(clase)

    # Nombre
    if d_nombre["Caracteres"] > 25 or d_nombre["Caracteres"] == 0 or  d_nombre["Palabras"] < 2:
        print("Nombre: Mal")
    else:
        print("Nombre: Ok")
    # Fecha
    if d_fecha["Caracteres"] !=8:
        print("Fecha: Mal")
    else:
        print("Fecha: Ok")
    # Clase
    if d_class["Caracteres"] == 1:
        print("Clase: Ok")
    else:
        print("Clase: Mal")

    for block in blocks:
        linea_pregunta = detectar_linea_pregunta(block)
        detectar_respuesta(block, linea_pregunta)
    respuestas_correctas = ['C', 'B', 'A', 'D', 'B', 'B', 'A', 'B', 'D', 'D']
    nuevo_orden = [0, 2, 4, 6, 8, 1, 3, 5, 7, 9]
    # Reorganizamos la lista de las respuestas de cada examen
    respuestas_correctas_ordenadas = [rta_examen[i] for i in nuevo_orden]
    contador = 0
    for i in range(len(respuestas_correctas)):
        if respuestas_correctas[i] == respuestas_correctas_ordenadas[i]:
            print(f'Pregunta {(i+1)}:', ' OK')
            contador += 1
        else:
            print(f'Pregunta {(i+1)}:', ' MAL')
    if contador >=6:
        condiciones.append('Aprobado')
    else:
        condiciones.append('Reprobado')
    print('Puntaje: ',contador,'/10')
generar_imagen_resultados(condiciones, nombres)
```

Fragmento de código que implementa la corrección automática de los exámenes, detectando las líneas verticales y horizontales que delimitan las preguntas y campos de encabezado (nombre, fecha y clase). El código corta estos campos para validarlos.

Luego, se analizan los contornos de los caracteres en cada campo para verificar si el nombre tiene al menos dos palabras, si la fecha contiene exactamente 8 caracteres, y si la clase es un único carácter.

Finalmente, se reorganizan las respuestas del examen según el orden correcto, y se comparan con las respuestas del modelo. Si al menos 6 respuestas son correctas, el examen es aprobado; en caso contrario, es desaprobado.

Además almacenando en una lista 'nombres' los crops de el apartado nombre en el encabezado de cada examen para una posterior visualización de este con la condición de cada alumno

---

```python
def generar_imagen_resultados(condiciones, nombres):
    """
    Esta funcion es la encargada de generar una imagen con los resultados de los examenes, recibe una lista con las condiciones de los examenes
    y una lista con los nombres de los alumnos y genera una imagen con los resultados de los examenes
    """

    # Parámetros del texto de para los resultados
    fuente = cv2.FONT_HERSHEY_SIMPLEX
    escala = 0.7
    grosor = 2
    espacio_entre_lineas = 10  # Espacio entre cada nombre

    ancho_max_nombres = max(nombre.shape[1] for nombre in nombres)
    alto_total = sum(nombre.shape[0] + espacio_entre_lineas for nombre in nombres)

    ancho_total = ancho_max_nombres + 250  # Espacio extra para el texto
    alto_total += 100  # Margen inferior

    img_resultado = np.ones((alto_total, ancho_total, 3), dtype=np.uint8) * 255  # Imagen blanca

    for i, nombre_img in enumerate(nombres):
        alto_nombre, ancho_nombre = nombre_img.shape[:2]   # Seleccionamos solo el alto y ancho de la imagen (Obiando los canales)
        img_resultado[eje_y:eje_y+alto_nombre, 50: 50+ancho_nombre] = cv2.cvtColor(nombre_img, cv2.COLOR_GRAY2BGR) # En el eje x definimos 50 de

        # Agregamos el texto
        condicion = condiciones[i]
        texto = f"{condicion}"


        if condicion == "Aprobado":
            color_texto = (0, 255, 0)
        else:
            color_texto = (0, 0, 255)

        #Definimos las dimensiones del texto en base a parametros de los nombres y una suma de pixeles predefinida
        cv2.putText(img_resultado, texto, (ancho_nombre + 70, eje_y + int(alto_nombre / 2) + 10), fuente, escala, color_texto, grosor)

        # Desplazar eje y para la siguiente linea
        eje_y += alto_nombre + espacio_entre_lineas

    cv2.imwrite("resultados_examenes.png", img_resultado)

    plt.imshow(cv2.cvtColor(img_resultado, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
```

Esta función es la encargada de generar una imagen con los resultados de los exámenes, recibe una lista con las condiciones de los exámenes y una lista con los nombres de los alumnos y genera una imagen con los resultados de los exámenes.

Imagen final que muestra los recortes de los nombres de los alumnos con los resultados del examen. Los recortes de los alumnos que aprobaron están destacados en verde, mientras que los desaprobados están en rojo, facilitando la visualización de los resultados