Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento de Imágenes 1 (PDI)

Informe Trabajo Práctico N°1

Integrantes del Grupo 3:

Sol Kidonakis

Claudia Leguiza

Betsabé Gómez

En este informe, exponemos el desarrollo llevado a cabo durante la realización del primer Trabajo Práctico de la materia. Nuestro primer objetivo fue abordar el problema 1, el cual nos desafió a implementar una técnica de ecualización local de histograma en una imagen específica para revelar elementos que estaban ocultos a simple vista. Posteriormente, nos sumergimos en el problema 2, que implicaba la corrección de exámenes de opción múltiple, la validación de varios campos (como nombres, ID código y día) y la generación de una imagen que diferencie entre los alumnos aprobados y reprobados.

A lo largo de este informe, ofrecemos una visión detallada del código utilizado, presentamos gráficos para una mejor comprensión y discutimos las dificultades que surgieron durante el proceso. Finalmente, extraemos conclusiones significativas a partir de nuestras experiencias en este proyecto.

Análisis y Desarrollo del Problema 1 - Ecualización Local de Histograma

Descripción del Problema

El problema 1 se centra en la implementación de la técnica de ecualización local de histograma en una imagen. Se requiere desarrollar una función que reciba como parámetros la imagen a procesar y el tamaño de la ventana de procesamiento, y que aplique la ecualización local del histograma a cada región de la imagen, moviendo la ventana de pixel a pixel.

Método Propuesto

Para abordar este problema, proponemos el siguiente método:

1. Preparación de la Imagen: Se agregarán bordes a la imagen para manejar los píxeles cercanos a los bordes de manera adecuada, evitando problemas de desbordamiento al acceder a píxeles fuera de los límites de la imagen original.

```
# Se agregan bordes para manejar los píxeles cercanos a los bordes de la image
image_with_borders = cv2.copyMakeBorder(image,
                                        half_window,
                                        half_window,
                                        half_window,
                                        half_window,
                                        cv2.BORDER_REPLICATE)
```

2. Iteración sobre los Píxeles: Se iterará sobre los píxeles de la imagen original, excluyendo los píxeles cercanos a los bordes que han sido añadidos para la preparación.

3. Extracción de la Ventana: En cada iteración, se extraerá una ventana alrededor del píxel actual, con un tamaño determinado por el parámetro del tamaño de la ventana.

```
# Itera sobre los píxeles de la imagen original
for i in range(half_window, height + half_window):
    for j in range(half_window, width + half_window):
        # Se extrae la ventana alrededor del píxel actual
        window = image_with_borders[i-half_window:i+half_window+1, j-half_wind
        # Aplica la ecualización de histograma a la ventana
        equalized_window = cv2.equalizeHist(window)
        # Guarda el píxel central de la ventana ecualizada
        result_image[i-half_window, j-half_window] = equalized_window[half_wir

return result_image
```

4. Ecualización de Histograma: Se aplicará la ecualización de histograma a la ventana extraída utilizando la función `cv2.equalizeHist` de OpenCV.

```
# Aplica la ecualización de histograma local con diferentes tamaños de ventana
img1 = apply_local_histogram_equalization(imagen_original, 3*3)
img2 = apply_local_histogram_equalization(imagen_original, 11*3)
img3 = apply_local_histogram_equalization(imagen_original, 33*3)
```

5. Guardado del Píxel Ecualizado: Se guardará el píxel central de la ventana ecualizada en la matriz de resultado.
6. Desplazamiento de la Ventana: Se desplazará la ventana un pixel hacia el costado y se repetirá el procedimiento hasta recorrer toda la imagen.

Análisis de Resultados y Influencia del Tamaño de la Ventana

Se analizará la imagen proporcionada (Figura 1) para identificar los detalles escondidos en diferentes zonas de la misma. Además, se evaluará la influencia del tamaño de la ventana en los resultados obtenidos, comparando los resultados obtenidos con diferentes tamaños de ventana.

Implementación y Evaluación

Finalmente, se implementará la función propuesta y se evaluará su desempeño en la imagen proporcionada, así como en otras imágenes de prueba. Se realizará una evaluación detallada de los resultados obtenidos y se discutirá la influencia del tamaño de la ventana en la calidad de la ecualización local de histograma.

Análisis y Desarrollo del Problema 2 - Corrección de Multiple Choice

Descripción del Problema

El problema 2 plantea el desafío de desarrollar un algoritmo capaz de corregir automáticamente exámenes tipo multiple choice. Para ello, se proporcionan imágenes de los exámenes resueltos en formato de imagen, y se espera que el script de Python identifique las respuestas correctas e incorrectas para cada pregunta y valide los datos del encabezado (nombre, ID, código y fecha). Además, se debe generar una imagen de salida que muestre qué alumnos han aprobado el examen y cuáles no.

Método Propuesto

Para abordar este problema de manera efectiva y precisa, se propone el siguiente método:

1. Preprocesamiento de las Imágenes:
    - Se carga cada imagen del examen y se convierte a escala de grises utilizando OpenCV (`cv2.imread()` y `cv2.cvtColor()`). Esto facilita el procesamiento posterior de las imágenes.
2. Validación de los Campos del Encabezado y Separación del Multiple Choice:

- Se desarrolla una función (`procesar_form()`), dicha función comienza calculando la suma de los valores en cada fila de la imagen binarizada. Esto proporciona una medida de la densidad de píxeles activos (blancos) en cada fila. Luego, se filtran las filas que tienen una densidad de píxeles activos mayor a 500, lo que ayuda a eliminar filas vacías o de ruido. A continuación, se obtienen los índices de las filas que pasan el filtro de densidad. Estos índices indican las posiciones verticales donde comienza y termina el encabezado del examen.
- Después, se recorren los índices de las filas válidas para detectar el inicio y el final del encabezado del examen. Se busca un espacio vertical suficientemente grande entre las filas para determinar el límite entre el encabezado y el cuerpo del examen. Una vez identificados los límites del encabezado, se segmentan las subimágenes correspondientes a los campos de nombre, ID, tipo y fecha dentro del encabezado del examen.
- Posteriormente, se identifican las columnas válidas dentro de cada fila para delimitar los campos de nombre, ID, tipo y fecha del encabezado del examen. Se extraen las subimágenes correspondientes a los campos de nombre, ID, tipo, fecha y múltiple choice utilizando los índices obtenidos en los pasos anteriores. Finalmente, se devuelven las subimágenes correspondientes a cada campo identificado en el encabezado del examen, así como la subimagen que contiene las preguntas de múltiple choice para su posterior procesamiento.

```python
def procesar_form(imagen_bool):
    img_rows = np.sum(imagen_bool, axis=1)
    filas_validas = img_rows > 500
    indices_validos = np.where(filas_validas)[0]
    umbral_espacio = 22

    primer_espacio_index = next((i for i in range(1, len(indices_validos)) if indices_validos[i] - indices_validos[i - 1] > umbral_espacio
    if primer_espacio_index is None:
        return None, None, None, None, None

    indice_inicial = indices_validos[primer_espacio_index - 1] + 1
    indice_final = indices_validos[primer_espacio_index] - 1

    img_final = imagen_bool[indice_inicial:indice_final]

    img_cols = np.sum(img_final, axis=0)
    umbral_columna_valida = 20
    columnas_validas = img_cols > umbral_columna_valida

    inicio_sub_imgs = []
    fin_sub_imgs = []

    i = 0
    while i < len(columnas_validas):
        if not columnas_validas[i]:
```

3. Conteo de Componentes Conectados:
    - Se implementa una función (`contar_componentes_conectados()`). Se utiliza la función `connectedComponentsWithStats` de OpenCV para identificar los componentes conectados en la imagen binarizada. Cada píxel se etiqueta con un número entero que indica a qué componente conectado pertenece.
    - Conteo de caracteres y palabras: Se inicializan las variables `caracteres` y `palabras` a 0. Si el número de componentes conectados es menor o igual a 2, se asume que no hay componentes conectados o solo hay uno que representa el fondo, por lo que tanto el número de caracteres como de palabras se establecen en 0. Para imágenes con más de dos componentes conectados, se ordenan las estadísticas de los componentes conectados por la coordenada x de su región. Luego, se calcula la distancia entre el borde izquierdo de cada componente conectado y el borde derecho del componente conectado anterior para determinar el espacio entre palabras. Si la distancia es mayor o igual a 5, se considera que hay un espacio entre palabras. Se cuentan los espacios y se suman 1 para obtener el número total de palabras. El número total de caracteres se calcula sumando el número de componentes conectados y restando el número de espacios, ya que cada componente conectado representa un carácter, pero no todos los componentes conectados son caracteres individuales.
    - Retorno de resultados: Se devuelve el número de caracteres y palabras calculados.

```python
def contar_componentes_conectados(img):
    f_point = img.astype(np.uint8)
    connectivity = 8
    num_labels, _, stats, _ = cv2.connectedComponentsWithStats(f_point, connectivity, cv2.CV_32S)
    caracteres = 0
    palabras = 0
    if num_labels <= 2:
        caracteres = num_labels - 1
        palabras = num_labels - 1
    else:
        ind_ord = np.argsort(stats[:,0])
        stats_ord = stats[ind_ord]
        resultados = []
        for i in range(2, num_labels):
            fila_actual = stats_ord[i]
            fila_anterior = stats_ord[i - 1]
            suma = fila_actual[0] - (fila_anterior[0] + fila_anterior[2])
            resultados.append(suma)
        area_promedio = np.mean(stats_ord[:, 4])  # Calculamos el área promedio de los componentes
        umbral = 3 * area_promedio  # Usamos un múltiplo del área promedio como umbral
        espacios = sum(1 for valor in resultados if valor >= umbral)
        palabras = espacios + 1
        caracteres = num_labels + espacios - 1
    return caracteres, palabras
```

4. Validar Campos (`validar_campos(examen)`):
    - Esta función valida los campos del encabezado del examen, incluyendo nombre, ID, código y fecha, utilizando la información sobre componentes conectados obtenida previamente.

```python
def validar_campos(examen):
    nombre, legajo, tipo, dia, multiple_choice = procesar_form(examen)

    def validar_componente(componente, caracteres_requeridos, palabras_requeridas=None):
        caracteres, palabras = contar_componentes_conectados(componente)
        if palabras_requeridas is None:
            resultado = "OK" if caracteres == caracteres_requeridos else "MAL"
        else:
            resultado = "OK" if caracteres == caracteres_requeridos and palabras == palabras_requeridas else "MAL"
        return resultado

    diccionario_validacion = {
        "Nombre y Apellido": validar_componente(nombre, caracteres_requeridos=(1, 25), palabras_requeridas=2),
        "Legajo": validar_componente(legajo, caracteres_requeridos=8, palabras_requeridas=1),
        "Código": validar_componente(tipo, caracteres_requeridos=1),
        "Fecha": validar_componente(dia, caracteres_requeridos=8, palabras_requeridas=1)
    }

    return diccionario_validacion, multiple_choice, nombre
```

5. Identificación de Respuestas Correctas e Incorrectas:
    - Se desarrolla la función (`respuestas()`). Este calcula la suma de píxeles en cada fila de la imagen binaria de las opciones de respuesta.

- Identifica las filas que contienen opciones de respuesta.
- Procesa cada grupo de filas para extraer las opciones de respuesta marcadas.
- Utiliza la transformada de Hough circular para detectar círculos que representan opciones de respuesta marcadas.
- Verifica si al menos el 50% de la vecindad de cada círculo está rellena de blanco.
- Almacena las preguntas y las respuestas seleccionadas en un diccionario.
- Retorna el diccionario de preguntas y respuestas seleccionadas.

```
def respuestas(multiple_choice_binario):
    respuestas = []  # Lista para almacenar las respuestas encontradas
    preguntas = []   # Lista para almacenar los números de pregunta correspondientes a las respuestas encontradas
    indices_renglones = procesar_renglones(multiple_choice_binario)  # Obtiene los rangos de filas válidas
```

6. Corrección de los Exámenes:
    - Se crea una función (`corregir_examen()`) para corregir los exámenes en función de las respuestas identificadas como correctas o incorrectas. Define un diccionario `respuestas_correctas` que contiene las respuestas correctas para cada pregunta del examen.
    - Itera sobre las respuestas proporcionadas por el estudiante y compara cada una con la respuesta correcta correspondiente.
    - Registra el resultado de la corrección para cada pregunta en un diccionario `correcciones`, donde las respuestas correctas se marcan como "OK" y las incorrectas como "MAL".
    - Calcula el número total de respuestas correctas y determina si el estudiante aprobó el examen (al menos 20 respuestas correctas).
    - Retorna el diccionario de correcciones y un valor booleano que indica si el estudiante aprobó el examen.

```
def corregir_examen(respuestas):
    respuestas_correctas = {
        1: 1, 2: 1, 3: 2, 4: 1, 5: 4, 6: 2, 7: 2, 8: 3, 9: 2, 10: 1, 11: 4,
        12: 1, 13: 3, 14: 3, 15: 4, 16: 2, 17: 1, 18: 3, 19: 3, 20: 4, 21: 2,
        22: 1, 23: 3, 24: 3, 25: 3
    }

    correcciones = {}
    contador_ok = 0
```

7. Generación de la Imagen de Salida:
    - Se genera una imagen de salida que muestra qué alumnos han aprobado y cuáles no, resaltando el nombre en verde o rojo según corresponda. Esto se logra mediante la manipulación de matrices NumPy y la generación de una imagen en escala de grises.

Conclusiones y Observaciones

El algoritmo desarrollado para el problema 1 se enfoca en mejorar la visualización de imágenes mediante la ecualización local del histograma, resaltando detalles específicos en diferentes áreas de la imagen.

El algoritmo desarrollado para el problema 2 proporciona una solución automatizada y eficiente para corregir exámenes multiple choice, minimizando la intervención manual y acelerando el proceso de evaluación.

En el proceso de búsqueda de soluciones, hemos optado por probar diferentes enfoques y corregir nuestros errores conforme avanzamos. Esta forma de trabajar nos ha permitido aprender de nuestros errores y acercarnos gradualmente a la solución que buscamos.