FACULTAD DE CIENCIAS EXACTAS, INGENIERIA Y AGRIMENSURA
UNIVERSIDAD NACIONAL DE ROSARIO

PROCESAMIENTO DE IMÁGENES I

Maximiliano Romano R-4634/5 – Facundo Molina M-7128/5

1 - **Ecualización local de Histograma:**

El objetivo de este ejercicio fue implementar una función de Ecualización Local del Histograma, para mejorar el contraste en imágenes donde los detalles de interés están ocultos debido a tener niveles de intensidad muy similares al fondo.

Este método utiliza un único histograma para toda la imagen, esta técnica opera en un vecindario (ventana) móvil. En cada paso:

1. Se define una ventana de tamaño M×N centrada en un píxel.
2. Se calcula el histograma solo con los píxeles dentro de esa ventana.
3. Se utiliza la función de distribución acumulada (CDF) de ese histograma local para mapear y transformar únicamente el valor del píxel central.
4. La ventana se desplaza y el proceso se repite.

La finalidad práctica fue aplicar la función a la imagen dada para encontrar los detalles escondidos y, además analizar la influencia del tamaño de la ventana(M×N) en el resultado final. Analizando en cada iteración un pixel de la imagen para determinar su vecindario completo y procesarlo.

- Problema de Acceso al Vecindario (Bordes)

```
if len(imagen.shape) > 2: # validación
    raise ValueError("La imagen debe estar en escala de grises")

M, N = tamaño_ventana # desempaqueta
imagen_padding = cv2.copyMakeBorder(imagen, M//2, M//2, N//2, N//2, borderType=cv2.BORDER_REPLICATE) # padding (agrega pixeles a la imagen con e
ceros_imagen = np.zeros_like(imagen) # llenado con 0

filas, columnas = imagen.shape # tamaño de la imagen para recorrer la misma
```

Al centrar una ventana M×N en un píxel cerca del borde de la imagen, el vecindario se extiende fuera de los límites.

- Técnica: Padding (Relleno con el valor del pixel cercano de borde).
- Implementación: Se utilizó cv2.copyMakeBorder() con el tipo cv2.BORDER_REPLICATE. Este método copia el valor de los píxeles de borde al área de relleno. El tamaño del padding (M/2y N/2) está diseñado para que, cuando la ventana de tamaño M×N se centre en cualquier píxel de la imagen original, toda la ventana siga estando dentro de los límites de la imagen con padding.
- Validación para que la imagen ingresada sea en escala de grises.

- Se inicializa imagen llenadas con zero.

  – Implementación

```
for i in range(filas):
    for j in range(columnas):
        ventana_local = imagen_padding[i:i+M, j:j+N] # crop
        hist = cv2.calcHist([ventana_local], [0], None, [256], [0,256]) #aplicamos histograma
        hist = hist.ravel() / hist.sum() # normalización (vector plano hist / cantidad de elementos (MxN))
```

Requiere, para cada píxel (i,j) de la imagen original, extraer una sub-matriz correspondiente a la imagen con padding.

- Técnica: Doble bucle for y slicing (Crop).
- Implementación: El bucle itera sobre las filas (filas) y columnas (columnas)(extraemos la cantidad de filas y colunas con shape) de la imagen original. Dentro del bucle, se accede a la sub-matriz de la imagen con padding.

```
    cdf = hist.cumsum() # distribución acumulada(256 elementos, de 0 a 1)
    cdf_normalizada = (cdf * 255).astype('uint8') # mapea la probabilidad acumulada a la gama de intensidades de salida, formato uint8

    ceros_imagen[i, j] = cdf_normalizada[imagen[i, j]] # transformación final

return ceros_imagen
```

- Cálculo

Para cada ventana, se debe calcular la función de mapeo que transformará el píxel central.

- Técnicas Utilizadas:
  1. Cálculo del Histograma (cv2.calcHist).
  2. Normalización a Función de Densidad de Probabilidad (PDF).
  3. Cálculo de la Función de Distribución Acumulada (CDF).

- Mapeo

El valor final del píxel (i,j) en la nueva imagen se obtiene utilizando el valor original de ese píxel como índice en la CDF normalizada.

- Técnica de Resolución: Mapeo por Índice y Conversión de Tipo.
- Implementación: La CDF (que va de 0 a 1) se escala a la gama de intensidades (0 a 255) y se convierte a uint8. El valor original del píxel imagen[i, j] se usa como índice para obtener el valor transformado de la CDF.

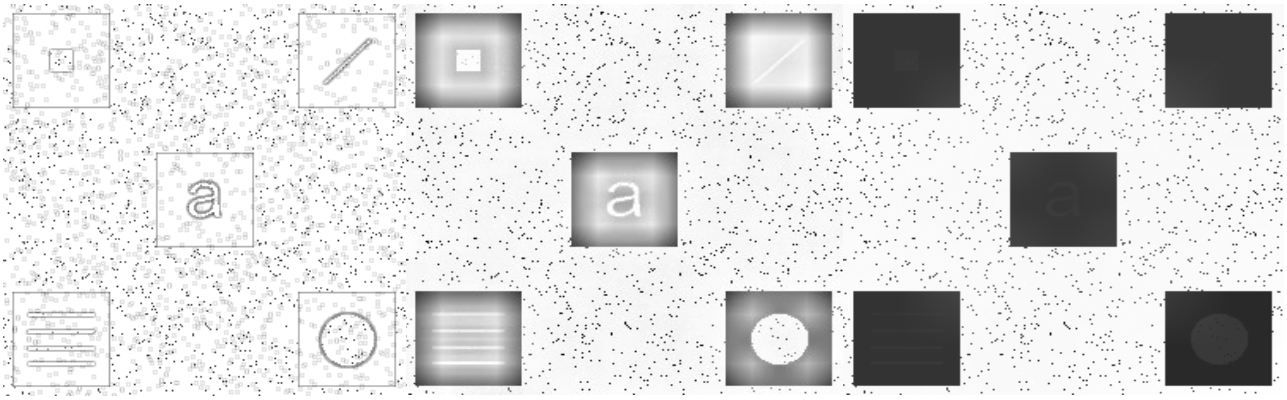


Ilustración 1: ventana 3x3 *Ilustración 2: ventana 30x30* *Ilustración 3: ventana 130x130*

El tamaño de la ventana (tamaño_ventana) es el parámetro crucial en este algoritmo, ya que define el área local sobre la cual se calcula el histograma, la función de distribución acumulada (CDF) y, por ende, la transformación de intensidad de cada píxel central.

1. Ventana chica (3×3)

- Imagen 1 (Gris con muchos detalles finos y ruido):
  - Efecto: Cada píxel se procesa basándose solo en sus vecinos más cercanos.
  - Resultado: alto contraste incluso en las regiones muy oscuras o muy claras. Sin embargo, esto también amplifica el ruido (puntos negros y cuadrados grises dispersos) al considerar las variaciones de intensidad locales como "detalles" importantes que deben contrastarse. El ruido de fondo y los contornos se vuelven muy distinguibles.

2. Ventana mediana (30×30)

- Imagen 2 (Gris con patrones claros y menos ruido):
  - Efecto: La ventana incluye una porción significativa del contexto alrededor de cada píxel.
  - Resultado: buen equilibrio entre el aumento de contraste y la baja del ruido. Los patrones escondidos (cuadrado, línea, 'a', rayas, círculo) son visibles y contrastados, pero el ruido de fondo se ha suavizado porque el histograma de la ventana ahora incluye muchos píxeles de fondo, diluyendo el efecto de los pocos píxeles de ruido dentro de esa área. El desenfoque alrededor de los patrones se debe a esta vecindad.

3. Ventana grande (130×130)

- Imagen 3 (Casi completamente oscura con patrones apenas visibles):
  - Efecto: La ventana es tan grande que prácticamente cubre una porción muy grande de la imagen para cada píxel. El histograma local de la ventana se vuelve muy similar al histograma global de toda la imagen.
  - Resultado: El resultado se acerca a una ecualización global. En una imagen con una distribución de intensidad sesgada (fondo ruidoso), el resultado es una imagen con un contraste general muy bajo y oscurecida. Los patrones se "esconden" nuevamente en la oscuridad, ya que el gran volumen de píxeles oscuros en la ventana domina el cálculo de la CDF.

- Resultados y Análisis de la Influencia de la Ventana: La imagen original Imagen_con_objetos_ocultos.tiff) presenta un bajo contraste general, donde los detalles se pierden en el fondo. Para la imagen de prueba, un tamaño chico (3×3) fue el que logró un mejor nivel de detalle, ya que permite la detección de estructuras muy finas que estaban contenidas en la misma, ventanas más grandes no logran ese nivel.
  - Realce de bordes y texturas: buena opción para realzar líneas muy delgadas, texturas chicas y cambios de intensidad grandes. Si los detalles ocultos son extremadamente pequeños (como las

marcas negras dispersas en su imagen de ejemplo), el 3×3 los destacará con el mayor contraste posible.

- Fidelidad Local: La transformación de contraste se adapta casi perfectamente a la variación de un píxel al siguiente, capturando la más mínima variación de intensidad
- Tiene la desventaja que realza el ruido, amplificando pequeñas variaciones(granulado).

Problema 2 - Validación de formulario

El objetivo de este ejercicio es implementar un sistema capaz de procesar automáticamente imágenes de formularios, extraer la información de ciertos campos (Nombre y Apellido, Edad, Mail, Legajo, Preguntas de selección y Comentarios) y validar su contenido según reglas. Adicionalmente, se generará una imagen de resumen y un archivo CSV con los resultados detallados de la validación.

Aplicamos un umbral fijo a la imagen en escala de grises. Todos los píxeles con intensidad superior al umbral se convierten a blanco (255) y los inferiores a negro (0). Esto transforma la imagen en un formato binario que facilita la detección de características y la identificación de caracteres.

```python
def umbralado_binario(imagen,n_umbral):
    umbral = n_umbral
    img_bin = (imagen > umbral) * 255 # convierte true a 255 y false a 0
    img_bin  = img_bin.astype(np.uint8) # aseguramos formato uint8
    #plt.imshow(img_bin,cmap='gray',vmin=0,vmax=255)
    return img_bin
```

Implementamos funciones para detectar líneas verticales y horizontales. Para cada fila o columna, se cuenta la cantidad de píxeles negros. Si este conteo excede un porcentaje mínimo del total de píxeles en esa dimensión, se considera que hay una línea.

```python
# DETECCIÓN DE LÍNEAS HORIZONTALES
def detectar_lineash(imagen, porcentaje_minimoh):

    lista_filas = []
    alto, ancho = imagen.shape[:2]
    umbral_minimoh = alto * porcentaje_minimoh

    for i in range(alto): # itera sobre cada columna
        conteo_pixeles_lineah = 0

        for j in range(ancho):  # itera sobre cada fila dentro de esa columna
            if imagen[i][j] == 0:  # detecta si son todos los pixeles de la col 0.
                conteo_pixeles_lineah += 1

        if conteo_pixeles_lineah >= umbral_minimoh:
            lista_filas.append(i)

    return lista_filas
```

```python
def detectar_lineasv(imagen, porcentaje_minimov):

    lista_columnas = []
    alto, ancho = imagen.shape[:2]
    umbral_minimov = alto * porcentaje_minimov

    for j in range(ancho): # itera sobre cada columna
        conteo_pixeles_lineav = 0

        for i in range(alto):  # itera sobre cada fila dentro de esa columna
            if imagen[i][j] == 0:  # detecta si son todos los pixeles de la col 0.
                conteo_pixeles_lineav += 1

        if conteo_pixeles_lineav >= umbral_minimov:
            lista_columnas.append(j)

    return lista_columnas
```

Extracción de Campos:

Detectamos las líneas del formulario y recortamos la imagen principal del contenido.

```python
ultima_linea_y_pos = img_bin.shape[0]
primer_linea = lineash_original["Linea_1"][1]

# crop para sacar encabezados
img_crop_form1 = img_bin[primer_linea:ultima_linea_y_pos,
                         lineasv_original["Linea_1"][0]:lineasv_original["Linea_2"][0]]
img_crop_form1 = img_crop_form1.astype(np.uint8)
```

Utilizando las coordenadas de las líneas detectadas, se recorta la imagen en sub-imágenes que corresponden a cada campo del formulario (Nombre y Apellido, Edad, Mail, Legajo, Preguntas, Comentarios).

```
# Recortes
nombre_y_apellido_crop = img_crop_form1[0 : lineash_crop_final["Linea_1"][0], :]

edad = img_crop_form1[lineash_crop_final["Linea_1"][1] : lineash_crop_final["Linea_2"][0], :]
mail = img_crop_form1[lineash_crop_final["Linea_2"][1] : lineash_crop_final["Linea_3"][0], :]
legajo = img_crop_form1[lineash_crop_final["Linea_3"][1] : lineash_crop_final["Linea_4"][0], :]
preg1 = img_crop_form1[lineash_crop_final["Linea_5"][1] : lineash_crop_final["Linea_6"][0], :]
preg2 = img_crop_form1[lineash_crop_final["Linea_6"][1] : lineash_crop_final["Linea_7"][0], :]
preg3 = img_crop_form1[lineash_crop_final["Linea_7"][1] : lineash_crop_final["Linea_8"][0], :]
comentarios = img_crop_form1[lineash_crop_final["Linea_8"][1] : lineash_crop_final["Linea_9"][0], :]
```

La imagen se invierte para que los caracteres se conviertan en objetos blancos sobre un fondo negro, lo cual es óptimo para cv2.connectedComponentsWithStats.

La función cv2.connectedComponentsWithStats identifica todos los grupos de píxeles conectados en la imagen. Cada componente se considera un caracter.

```
celda_invertida = 255 - celda_img
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(
    celda_invertida, 8, cv2.CV_32S
)
```

Se aplican filtros de área, altura y anchura de cada componente. Esto es crucial para eliminar ruido

La cantidad de componentes que superan los filtros se cuenta como el número total de caracteres.

Los componentes válidos se ordenan horizontalmente. Se calcula la distancia entre el final de un componente y el inicio del siguiente. Si esta distancia excede un umbral, se considera una separación entre palabras.

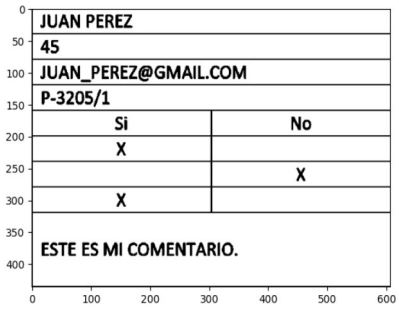
Validación de Contenido: Para cada campo, se compara la cantidad de caracteres y palabras con un conjunto de reglas de específicas :

```
def validar_nombre_apellido(caracteres, palabras):
    return "OK" if (palabras >= 2 and caracteres <= 25) else "MAL"

def validar_edad(caracteres, palabras):
    return "OK" if (2 <= caracteres <= 3 and palabras == 1) else "MAL"

def validar_mail(caracteres, palabras):
    return "OK" if (palabras == 1 and caracteres <= 25) else "MAL"

def validar_legajo(caracteres, palabras):
    return "OK" if (caracteres == 7 and palabras == 1) else "MAL"

def validar_comentarios(caracteres, palabras):
    return "OK" if (palabras >= 1 and caracteres <= 25) else "MAL"

def validar_pregunta(caracteres, palabras):
    return "OK" if (palabras == 1 and caracteres == 1) else "MAL"
```

Usamos la función procesar_formulario para aplicar todas las funciones a los formularios:

Cargamos la imagen y aplicamos el umbralado.

Detectamos las líneas del formulario y recortamos la imagen principal del contenido.



Recortamos la imagen de contenido en sub-imágenes para cada campo.

Para cada campo, llama a deteccion_elementos y luego a las funciónes de validación.

Agregamos los resultados de cada campo en un diccionario.

```python
# detección y validacion
nombre_info = deteccion_elementos(nombre_y_apellido_crop)
edad_info = deteccion_elementos(edad)
mail_info = deteccion_elementos(mail, UMBRAL_ESPACIO_FIJO=30)
legajo_info = deteccion_elementos(legajo, UMBRAL_ESPACIO_FIJO=30)
preg1_info = deteccion_elementos(preg1)
preg2_info = deteccion_elementos(preg2)
preg3_info = deteccion_elementos(preg3)
comentarios_info = deteccion_elementos(comentarios)


resultados_valid = {
    "Nombre y Apellido": validar_nombre_apellido(*nombre_info),
    "Edad": validar_edad(*edad_info),
    "Mail": validar_mail(*mail_info),
    "Legajo": validar_legajo(*legajo_info),
    "Pregunta 1": validar_pregunta(*preg1_info),
    "Pregunta 2": validar_pregunta(*preg2_info),
    "Pregunta 3": validar_pregunta(*preg3_info),
    "Comentarios": validar_comentarios(*comentarios_info),
}
```

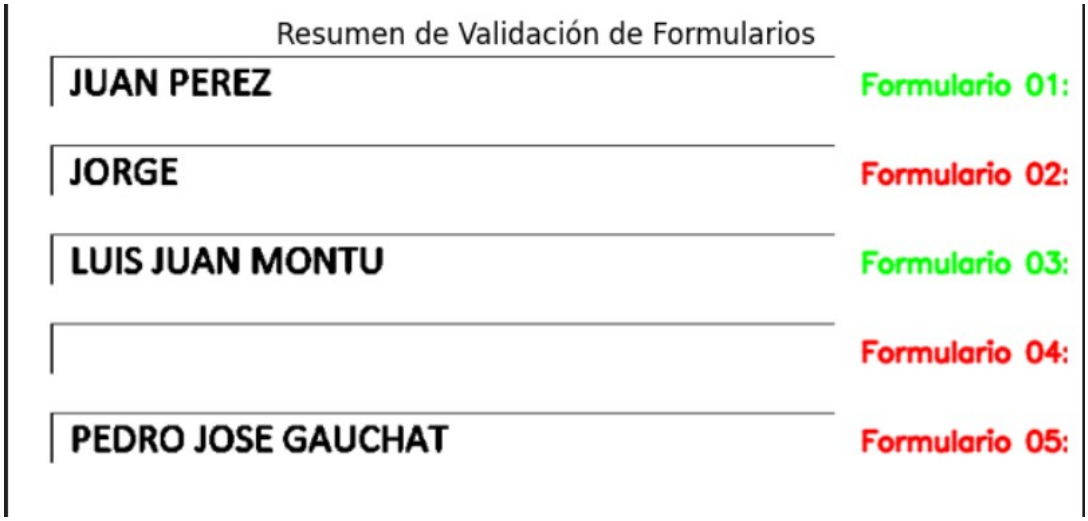
Determinamos el estado de validacion para el formulario completo, "OK" si todos los campos son "OK", de lo contrario "MAL".

```python
# Determinar si el formulario es globalmente correcto
formulario_es_correcto = all(estado == "OK" for estado in resultados_valid.values())
estado_global = "OK" if formulario_es_correcto else "MAL"
```

Devuelve el recorte del campo "Nombre y Apellido", el formulario y el diccionario con todos los resultados detallados.

```
Resultados de validación para formulario_01.png:
> Nombre y Apellido: OK
> Edad: OK
> Mail: OK
> Legajo: OK
> Pregunta 1: OK
> Pregunta 2: OK
> Pregunta 3: OK
> Comentarios: OK
Estado global del formulario: OK
```

Generamos una imagen de resumen que muestra el campo "Nombre y Apellido" de cada formulario procesado, acompañado de un indicador de color verde o rojo que señala si el formulario completo ha sido validado como "OK" o "MAL".



Generamos un archivo CSV que registra el resultado de la validación (OK/MAL) para cada campo de cada formulario.

```
ID,Nombre y Apellido,Edad,Mail,Legajo,Pregunta 1,Pregunta 2,Pregunta 3,Comentarios
1,OK,OK,OK,OK,OK,OK,OK,OK
2,MAL,MAL,OK,MAL,MAL,MAL,OK,MAL
3,OK,OK,OK,OK,OK,OK,OK,OK
4,MAL,MAL,OK,MAL,OK,MAL,MAL,MAL
5,OK,MAL,OK,OK,MAL,MAL,MAL,OK
```