# PROCESAMIENTO DE IMÁGENES

# 2023


1 OCTUBRE

**INTEGRANTES:**

**Ponce, Daniel**

**Previgliano, Marcos**

**Sanchez, Salvador**

**Yañez, Mirian**

# PROCESAMIENTO DIGITAL DE IMÁGENES I
# Trabajo Práctico Nº1

## PROBLEMA 1 – Ecualización local de histograma

Introducción:

En este problema se pide que busquemos figuras ocultas en la imagen que se nos ha proporcionado. Para este desafío, hemos decidido emplear la técnica de ecualización local del histograma. Esta estrategia nos ofrece la posibilidad de descubrir detalles que podrían escapar a métodos convencionales de procesamiento de imágenes, al aplicar la ecualización del histograma de manera específica a pequeñas ventanas móviles en la imagen.

Para ello, en este proceso, es imprescindible determinar el tamaño de ventana más efectivo. Este parámetro influye directamente en nuestra capacidad para identificar con claridad los objetos ocultos. Buscamos un equilibrio entre resaltar detalles finos y evitar el ruido innecesario en la imagen.

Para encontrar la configuración óptima, realizaremos varios ensayos, explorando diferentes tamaños de ventana y dando a conocer la información valiosa que obtengamos. Nuestra meta es descubrir la combinación perfecta que maximice la visualización de las figuras ocultas, proporcionando información valiosa sobre la distribución de intensidades en la imagen y cómo optimizar esta técnica para revelar detalles significativos, en este caso, las figuras ocultas.

Desarrollo:

Para resolver dicho problema, decidimos crear una clase llamada "**LocalHistogramEqualizer**", la cual recibe como parámetro la imagen y el tamaño de ventana. Luego, ejecutamos el método **show_images,** el cuál nos muestra la imagen previa a la ecualización y a su lado la imagen post ecualización.

Es interesante remarcar que el método que se encarga de aplicar la técnica es el método **equalize.**

A continuación mostramos cómo se utiliza el código:

```
window_size = int(input("Selecciona un tamaño de ventana para el Ejercicio 1: "))
image_path = 'Imagenes/Imagen_con_detalles_escondidos.tif'
equalizer = LocalHistogramEqualizer(image_path, window_size)
equalizer.show_images()
```

```python
class LocalHistogramEqualizer:
    """
    Clase para realizar la ecualización local del histograma en una imagen en escala de grises.

    Args:
      image_path (str): Ruta de la imagen de entrada.
      window_size (int): Tamaño de la ventana para la ecualización local.

    Attributes:
      image (numpy.ndarray): La imagen en escala de grises a procesar.
      window_size (int): El tamaño de la ventana para la ecualización local.
      logger (logging.Logger): Instancia del logger.
      output_image (numpy.ndarray): La imagen en escala de grises luego de procesar.
    """
    def __init__(self, image_path, window_size):
        self.logger = setup_logger()
        self.image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        self.window_size = window_size
        self.output_image = self.equalize()
```

Método que se encarga de realizar la ecualización:

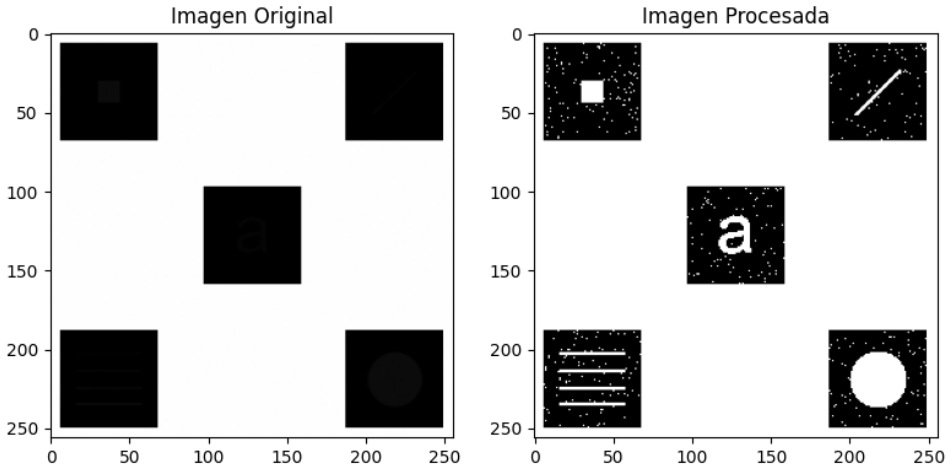
```python
def equalize(self):
    """
    Realiza la ecualización local del histograma en la imagen de entrada.
    Returns:
       numpy.ndarray: La imagen procesada con ecualización local del histograma, o None si ocurre un error.
    """
    try:
        height, width = self.image.shape[:2]
        half_window = self.window_size // 2
        output_image = np.copy(self.image)

        for y in range(half_window, height - half_window):
            for x in range(half_window, width - half_window):
                roi = self.image[y - half_window:y + half_window + 1, x - half_window:x + half_window + 1]
                hist, _ = np.histogram(roi.flatten(), bins=256, range=(0, 256))
                if hist.max() == 0:
                    continue
                cdf = hist.cumsum()
                if (cdf.max() - cdf.min()) == 0:
                    continue
                cdf_normalized = (cdf - cdf.min()) * 255 / (cdf.max() - cdf.min())
                output_image[y, x] = cdf_normalized[roi[0, 0]]

        return output_image
    except Exception as e:
        self.logger.error(f'Error en la ecualización local del histograma: {str(e)}')
        return None
```

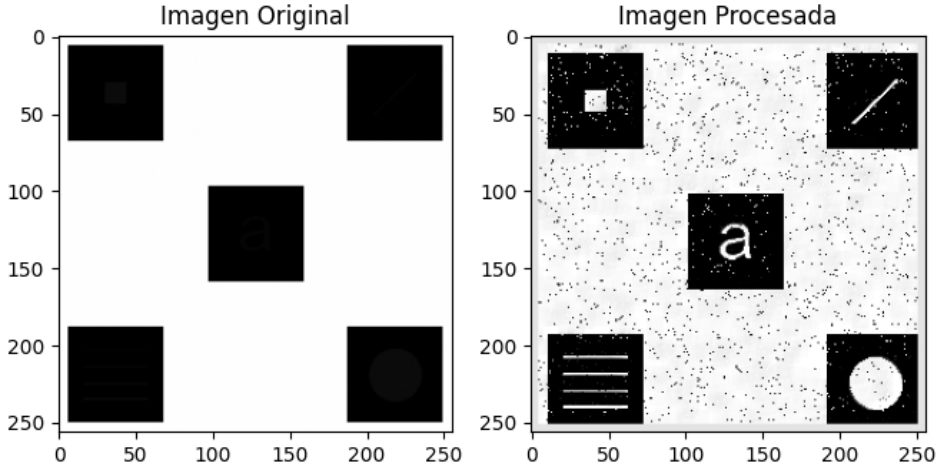
## Pruebas:

### Tamaño de ventana: 1



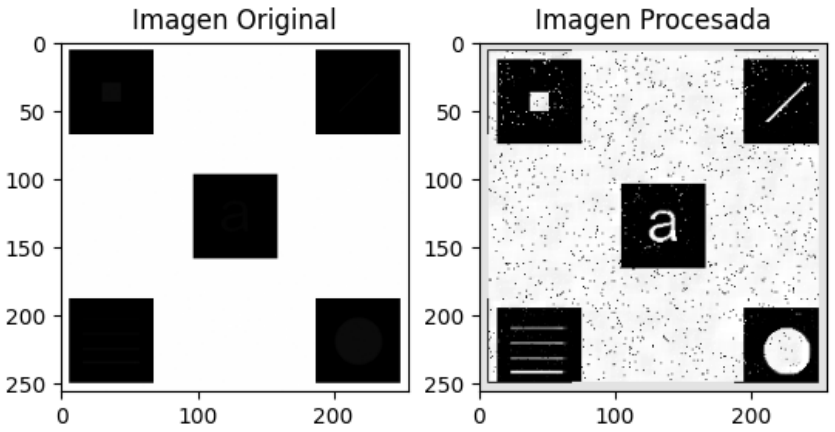

```
Menú:
1) Ecualización local de histograma
2) Validación de formulario
s) Salir
Selecciona una opción (1, 2, o s para salir): 1
Selecciona un tamaño de ventana para el Ejercicio 1: 1
```

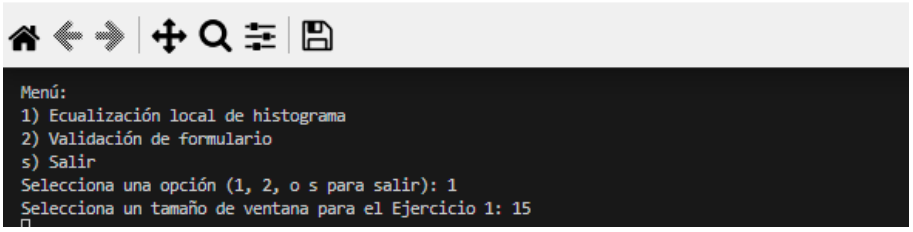
### Tamaño de ventana: 10



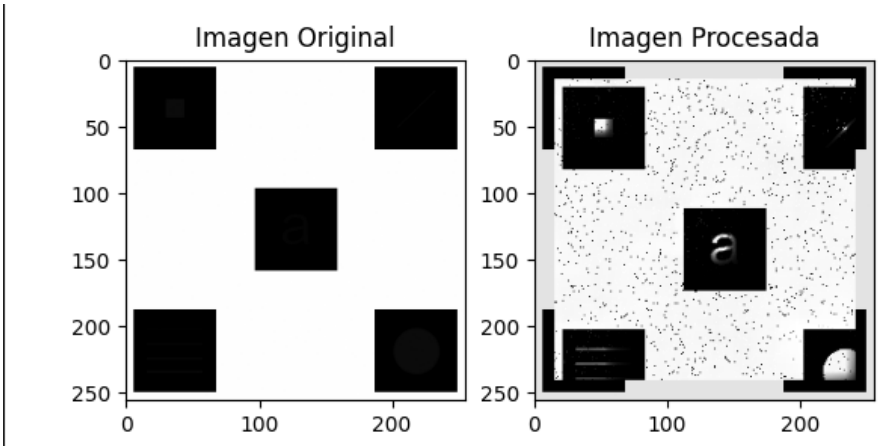

```
Menú:
1) Ecualización local de histograma
2) Validación de formulario
s) Salir
Selecciona una opción (1, 2, o s para salir): 1
Selecciona un tamaño de ventana para el Ejercicio 1: 10
```

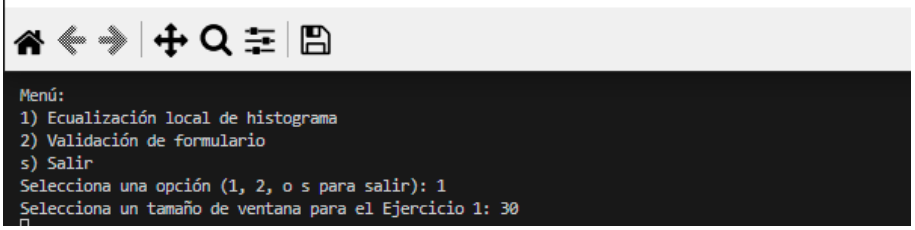
## Tamaño de ventana: 15



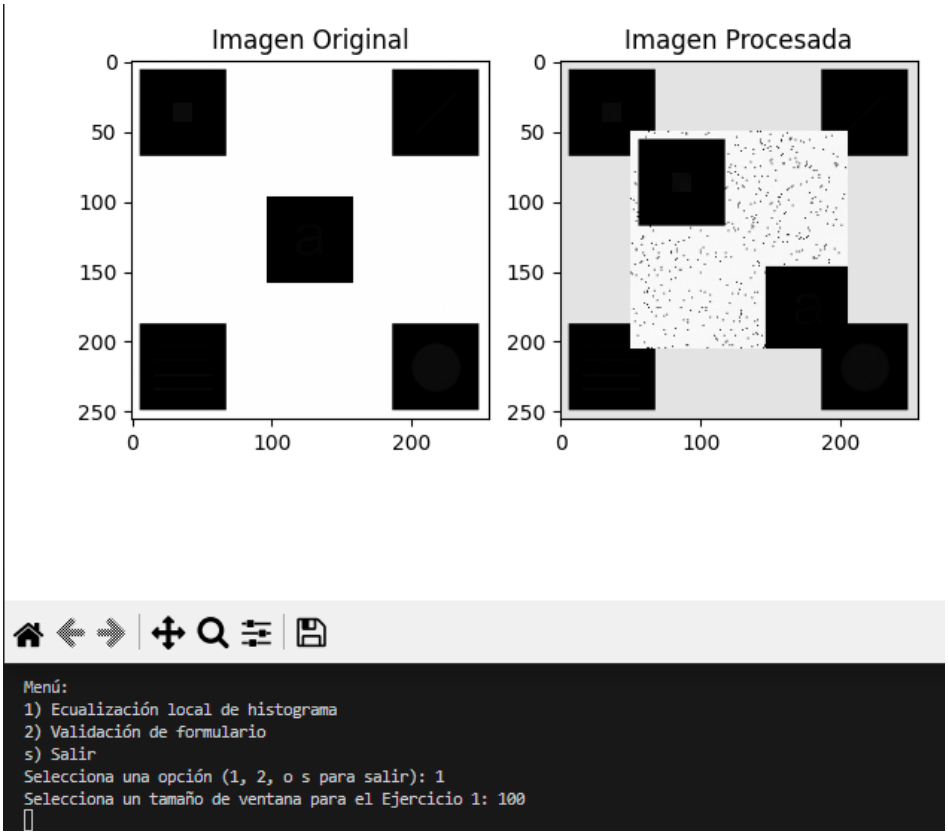




## Tamaño de ventana: 30



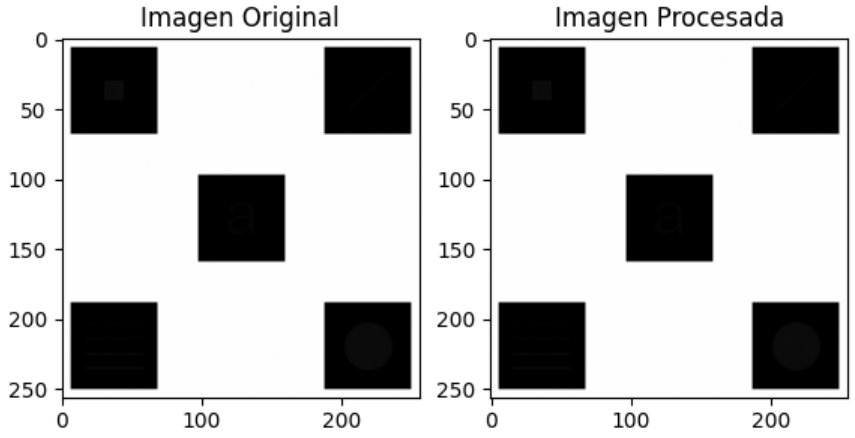



## Tamaño de ventana: 100




```
Menú:
1) Ecualización local de histograma
2) Validación de formulario
s) Salir
Selecciona una opción (1, 2, o s para salir): 1
Selecciona un tamaño de ventana para el Ejercicio 1: 100
```

## Tamaño de ventana: 256




```
Menú:
1) Ecualización local de histograma
2) Validación de formulario
s) Salir
Selecciona una opción (1, 2, o s para salir): 1
Selecciona un tamaño de ventana para el Ejercicio 1: 256
```

Conclusión:

Después de realizar varios ensayos con diferentes tamaños de ventana en la técnica de ecualización local del histograma, hemos llegado a la conclusión de que el tamaño de ventana ideal es 1. En este caso, notamos que no hay saturación excesiva y se logra una mejora en la visualización de las figuras ocultas.

Sin embargo, a medida que aumentamos el tamaño de la ventana, específicamente a partir de 10, comenzamos a enfrentar problemas. Se empieza a notar la aparición de ruido en la imagen, y a medida que la ventana crece aún más, no solo se intensifica el ruido, sino que también se observan distorsiones notables en la imagen.

## PROBLEMA 2 – Validación de formulario.

Introducción :

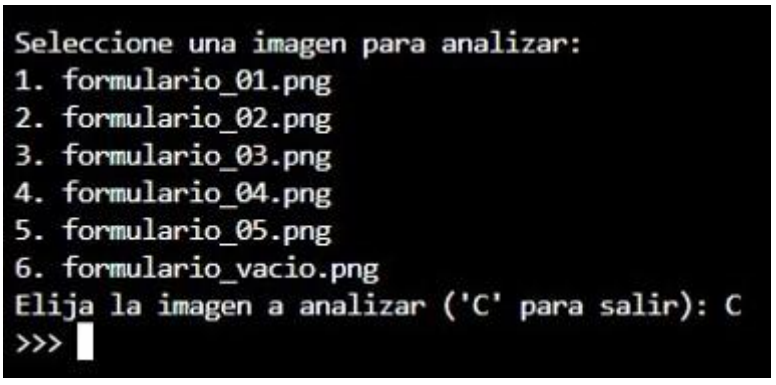
Extraer la información que se halla en las imágenes de los formularios.
Esta información se debe validar según el campo que corresponda y mostrar por pantalla su estado de validación.

Nombre y Apellido: debe contener al menos 2 palabras y no más de 25 caracteres.
Edad: debe contener 2 o 3 caracteres.
Mail: debe contener 1 palabra y no más de 25 caracteres.
Legajo: debe contener 1 palabra con 8 caracteres.
Preguntas 1 a 3: debe marcar con un carácter una de las dos celdas Si y No.
Comentario: no debe tener más de 25 caracteres.

Para la resolución dividimos el problema en dos etapas; la primera, segmentación y recorte de las imágenes, para obtener la subimagen que tiene la información y la segunda, clasificación de los elementos (letras y palabras) junto con su validación.

Desarrollo:

Primero debemos elegir que imagen de los formularios vamos a procesar.




Hecho esto, se procede a la detección de los renglones del formulario.

| 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. | |

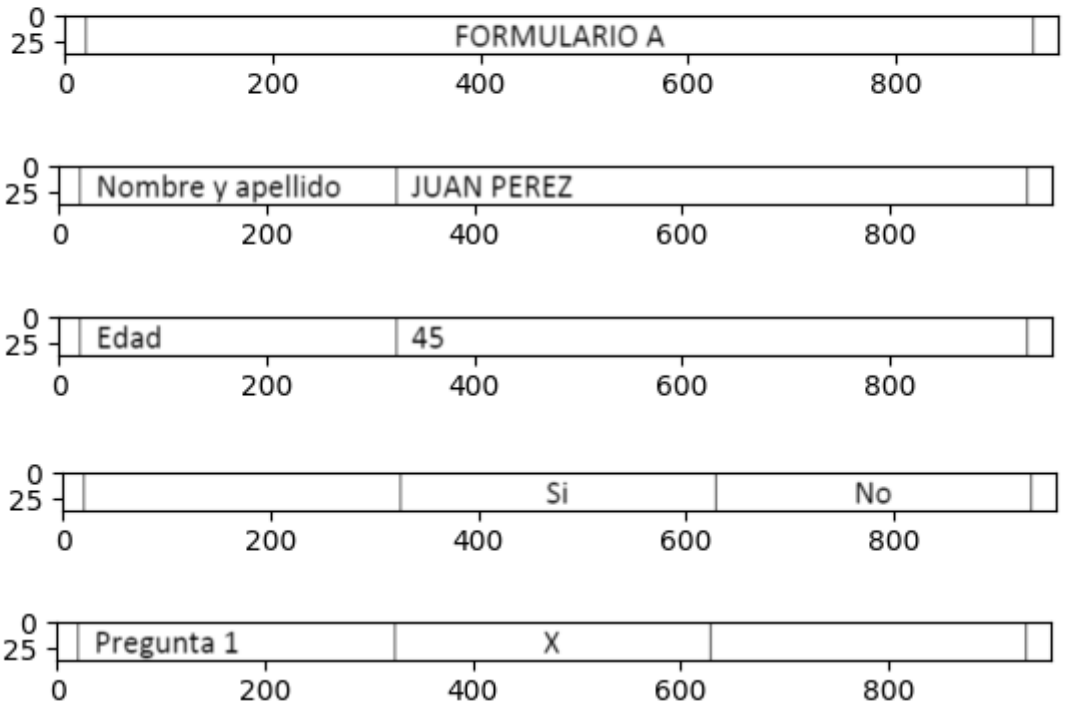
1) Segmentación y recorte de las imágenes

1.A) Segmentación vertical:

La detección se realiza al comparar los valores promedio de las filas de la imagen con un valor de umbral elegido rigurosamente. Cuando el valor promedio no alcanza el umbral, se guarda la posición de la fila. Acabamos de detectar una fila con un color más oscuro que el valor del umbral .

Como la imagen se encuentra en escala de grises debemos cerciorarnos que al cortar las subimágenes éstas no contengan restos del borde oscuro del renglón. Para ello sumamos 2 filas al borde inicial y, con el mismo criterio, quitamos 2 filas al borde final. Guardamos cada una de las subimágenes.

Recortes del formulario.




Eliminamos las subimágenes que no proporcionan información útil.

1.B) Segmentación verticales:

Realizamos de forma similar el proceso de detección de bordes verticales.

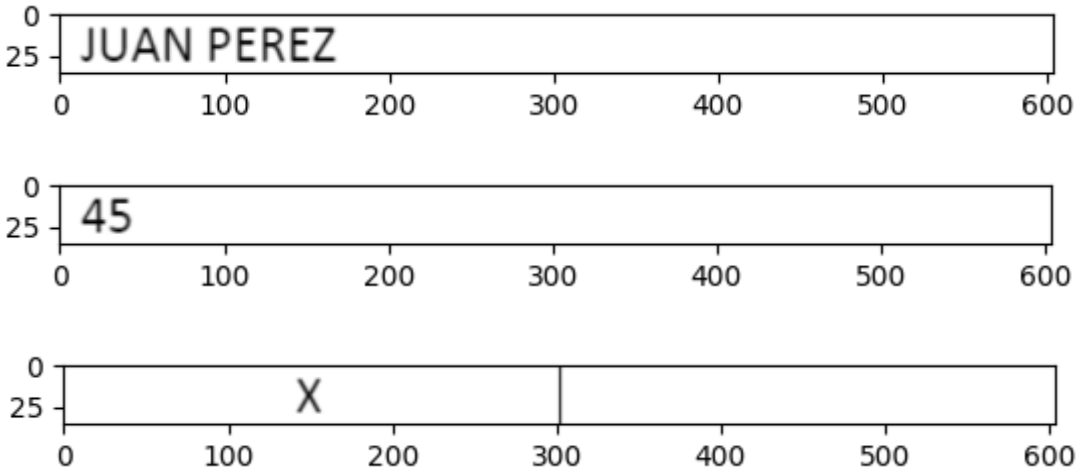
El vector de valores promedio de la imagen ahora tiene los valores promedio por columnas.

Recorremos las columnas en búsqueda de valores que se encuentren por debajo del umbral y en tal caso, guardamos la posición de la columna en una lista.

Luego tomamos estos valores guardados en la lista para generar los recortes de las subimágenes. Sabiendo que se debe descartar la posición inicial porque es el título del renglón (Nombre y Apellido, Edad, mail, etc), y la última posición ya que es un espacio vacío conservando el restante para su análisis.

Como estamos trabajando con una imagen en escala de grises hacemos el mismo tratamiento adicionando o sustrayendo 2 columnas según corresponda inicio o fin del recorte.

Recortes con la información:




2. Clasificación de los elementos y validación.

Recorremos cada una de las subimágenes del punto anterior utilizando la función enumerate.

Pasamos por parámetro cada una de las subimágenes a las función correspondiente para obtener el número de caracteres, función **count_caracter**(2.A), y la cantidad de palabras, función **count_word**(2.B).

Utilizamos el índice de estas subimágenes para chequear las características específicas de validación de cada campo a través de un if.

Los resultados se guardan en una lista que luego se muestra por pantalla.

```
1. Nombre:      MAL
2. Edad:        OK
3. Mail:        MAL
4. Legajo:      MAL
5. Pregunta 1:  OK
6. Pregunta 2:  MAL
7. Pregunta 3:  MAL
8. Comentarios: OK
0
Presione Enter para continuar...
```

2.A) Función count_caracter.

Esta función recibe una imagen y devuelve la cantidad de caracteres que hay en ella. Para ello convierte la imagen a una imagen binaria para poder hacer etiquetado de las componentes conectados. El tipo de conectividad que utilizamos es 4 conectada.

Imagen original

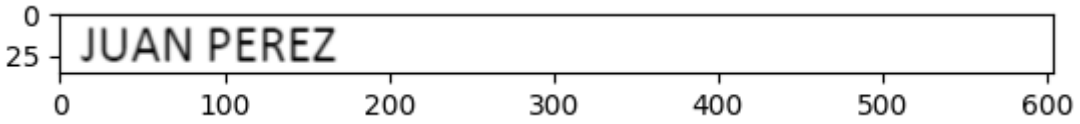



Imagen binarizada

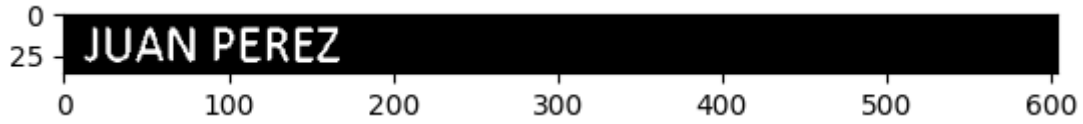
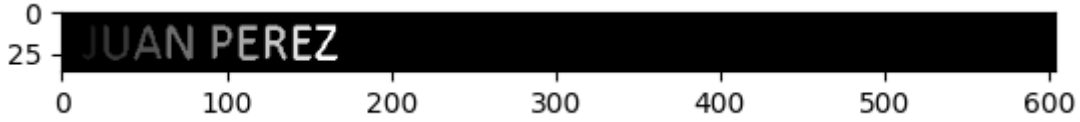



Imagen de componentes conectadas




Solo me interesa la cantidad de letras, antes de devolver el valor, debemos sustraer el fondo.

2.B) Función count_word.

Esta función recibe una imagen y devuelve la cantidad de palabras que contiene la imagen.

Convierte la imagen que recibe en binaria para buscar los contornos en la imagen obteniendo como resultado los puntos de cada elemento encontrado en la imagen.

Ordenamos los valores y recorremos los datos extraídos para calcular la distancia entre el final de la primera letra y el inicio de la siguiente. Si la distancia medida supera

el valor de umbral fijado entonces detectó una palabra, si ninguna superó la distancia estamos ante el caso de una única palabra.

Devuelvo el número de palabras detectadas.

Conclusión:

Durante la resolución pudimos observar la importancia del valor del umbral para la correcta segmentación de la imagen en escala de grises.

También al definir un objeto, cuando utilizamos el etiquetado de componentes conectados, es necesario prestar atención a la elección de los diferentes tipos de vecindarios. Al igual que al elegir el valor de la distancia entre palabra y letras ya que de no hacerlo de la manera correcta obtenemos valores erróneos en el análisis de las imágenes.