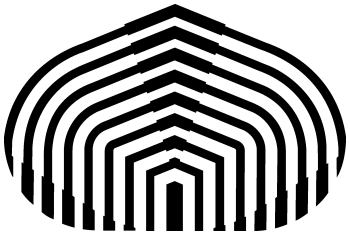

Universidad Simón Bolívar

EC5422 - Procesamiento Digital de Imágenes

Tarea 1: Detección de Caracteres

Leonardo Cabrera 13-10194
Gabriel Noya 13-10982

25/02/2018

Índice

Índice de figuras

1. Introducción

Se implementará un algoritmo de segmentación para la detección de texto en una imagen escaneada. El algoritmo deberá contar la cantidad de caracteres que se encuentren en la imagen, para esto se utilizará un algoritmo de segmentación para identificar los bordes de cada caracter y posteriormente contar cada uno de estos, trabajando además con imágenes binarias donde se puedan distinguir los caracteres de los demás elementos como el fondo.

Además, como contenido adicional coloco la opción de utilizar un algoritmo de reconocimiento de caracteres llamado OCR (Optical Character Recognition) por sus siglas en ingles, el cual se explicara en la ultima sección de este informe.

2. Comandos para el funcionamiento del programa

Para mejorar el funcionamiento del programa se implemento un modo de cambiar los parámetros según los argumentos que se ingresen al momento de ejecutar el programa por la consola, parámetros como el nombre de la imagen que se va a utilizar , el alto del kernel que se utiliza como elemento estructural y si se desea utilizar el algoritmo implementado por nosotros o el de OCR.

Para correr el programa por consola tomando los valores predeterminados (-n Cabrera-Noya.png -k 5), se ejecuta el siguiente comando:

```
> python tarea1.py
```

Si se desea cambiar el nombre del archivo de la imagen que se utilizará se añade el siguiente argumento a la configuración predeterminada, -n nombreDelArchivo, por ejemplo:

```
> python tarea1.py -n imagen.png
```

También se puede modificar la cantidad de filas del elemento estructural con el argumento -k seguido del alto del kernel. Esto es útil cuando el interlineado es muy pequeño y el elemento estructural une las líneas del texto ó cuando el tamaño de letra es muy grande y el espacio entre el punto de la 'i' (por ejemplo) con el resto del caracter es muy grande. Para esos casos, hay que disminuir y aumentar la altura del elemento estructural respectivamente. Esto se utiliza de la siguiente manera:

```
> python tarea1.py -k 5
```

Por último, podemos cambiar si se desea ejecutar el algoritmo implementado por nosotros (es la opción predeterminada) u OCR. La manera de utilizar OCR consiste en añadir el argumento -ocr, como se muestra a continuación:

```
> python tarea1.py -ocr
```

También podemos combinar los parámetros de la siguiente manera:

```
> python tarea1.py -n imagen.png -k 4
```

3. Detección de caracteres

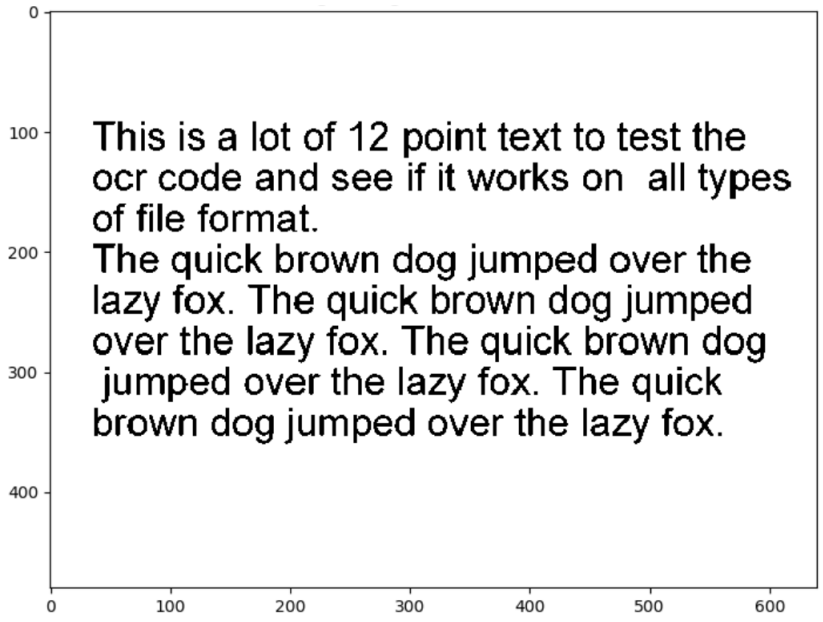

Figura 1: Imagen original.

Para detectar los caracteres debemos seguir una serie de pasos, listados a continuación:

- Filtrar el ruido
- Convertir la imagen a una imagen binaria
- Encontrar y contar los contornos

3.1. Filtrado de ruido

Para aplicar un filtro contra el ruido debemos aplicar la operación de dilatación y posteriormente la operación de erosión. Estas operaciones se realizaron con las funciones cv2.dilate() y cv2.erode(). Para nuestro caso utilizamos un kernel 2x2 y una sola iteración para cada operación.

Luego del filtrado, la imagen obtenida es la siguiente:

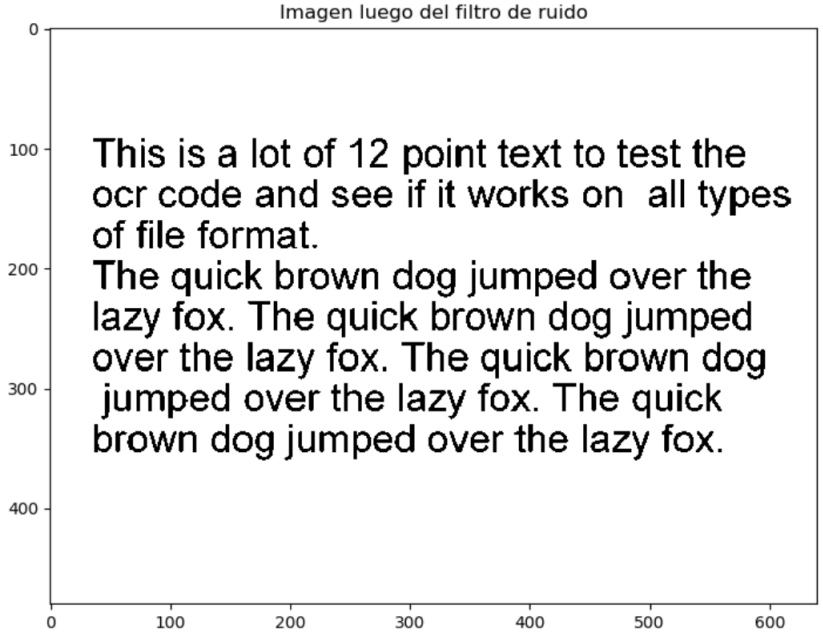

Figura 2: Imagen luego del filtro contra el ruido.

3.2. Imagen binaria

Para posteriormente utilizar la función cv2.findContours() debemos convertir la imagen a una imagen binaria con el fondo negro y los caracteres blancos. Utilizando la función cv2.adaptiveThreshold(), con un umbral binario e invertido, obtenemos la siguiente imagen:

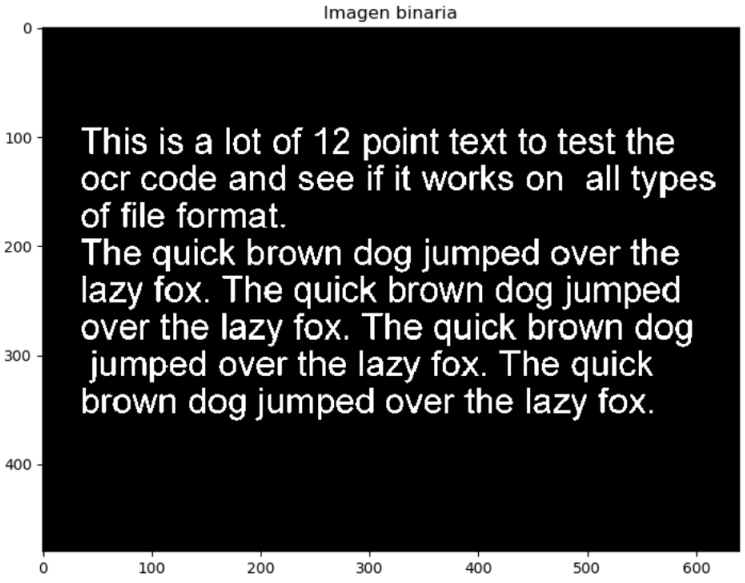

Figura 3: Imagen binaria.

3.3. Detección de contornos con filtro Canny

El filtro Canny utiliza la función gradiente y dos umbrales para detectar los contornos de una imagen. Este filtro se aplica con la función de OpenCV cv2.Canny(), que recibe la imagen y los dos umbrales a tomar en cuenta.

Al introducir nuestra imagen binaria en la función, esta nos retorna la siguiente imagen:

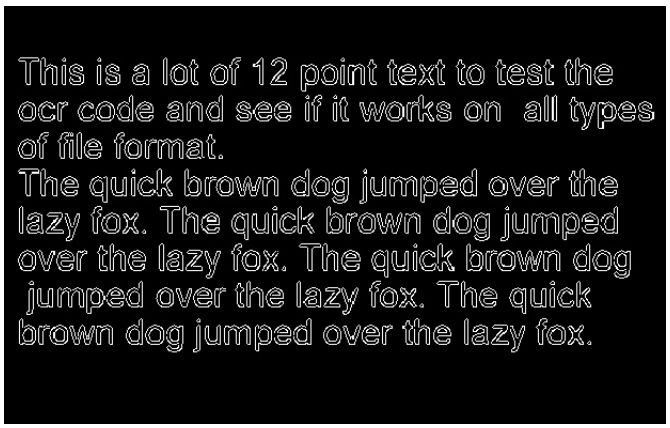

Figura 4: Imagen luego del filtro Canny.

3.4. Contornos

La función cv2.findContours() nos retorna la posición de cada punto de los contornos de cada caracter. Ya que no necesitamos todos los puntos, podemos utilizar el parámetro 'cv2.CHAIN_APPROX_SIMPLE' para que la función nos retorne únicamente cuatro puntos que rodean al contorno.

Existe un parámetro adicional RETR_XXXXXX, y esta se utiliza para relacionar los contornos entre si. Específicamente, este nos indica la jerarquía entre cada contorno, es decir, cual contorno se encuentra dentro de otro. Al utilizar el parámetro 'RETR_EXTERNAL', nos aseguramos que solo los contornos con mayor jerarquía se mantengan, es decir, solo los contornos externos se tomarán en cuenta. Este parámetro se utilizó ya que los caracteres con orificios internos como la 'P', 'B', 'a', retornaban contornos tanto para el caracter como para su orificio interno.

Para mostrar en la imagen los cuadros verdes, se utilizó la función cv2.rectangle(), utilizando los puntos obtenidos con la función cv2.findContours(). Al aplicar estos parámetros y funciones obtenemos la siguiente imagen:

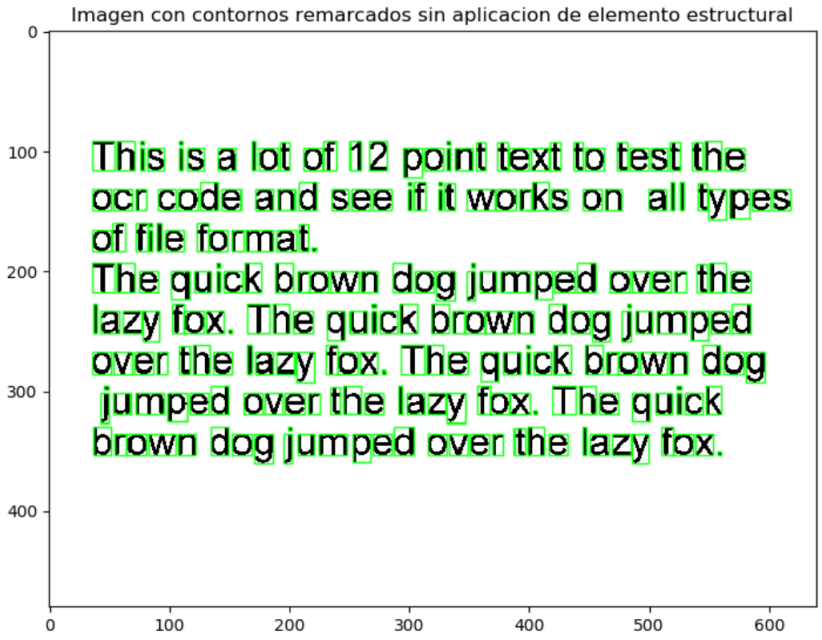

Figura 5: Imagen con contornos sin elemento estructural aplicado.

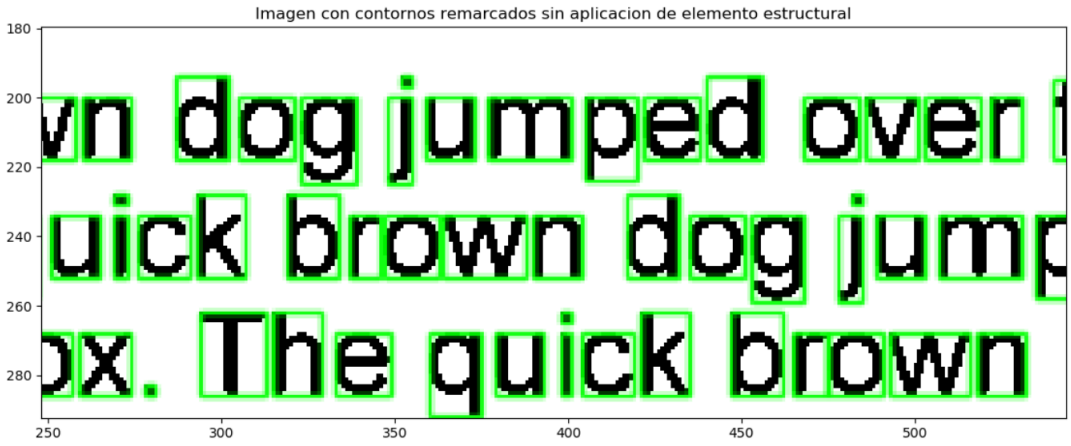

Figura 6: Imagen con contornos sin elemento estructural aplicado (ampliada).

Como podemos observar, las letras 'i' y 'j' se toman como dos caracteres distintos, y esto se debe a que el punto y el resto del caracter se encuentran separadas en su totalidad (no existe vecindad).

Para arreglar este problema, debemos barrer la imagen con un elemento estructural. Este proceso esta descrito en la siguiente sección.

3.5. Elemento estructural

La solución para el problema con los caracteres 'i' y 'j' es, de alguna manera, juntar los puntos con el resto del caracter. Para ello debemos aplicar un elemento estructural con forma (n, 1), siendo n el numero de filas del kernel,valor que se encuentra configurado por defecto en 5, pero ingresando el arguemnto -k seguido de un entero se puede modificar estas dimensiones, permitiendo así ajustarlo a letras de diferentes tamaños. Al hacer un barrido por la imagen con este elemento estructural, los píxeles en blanco (el espacio entre punto y el resto del caracter) se promediaran con los píxeles en negro (los del caracter) en dirección vertical.

Aplicando el barrido a la imagen binaria obtenemos la siguiente imagen:

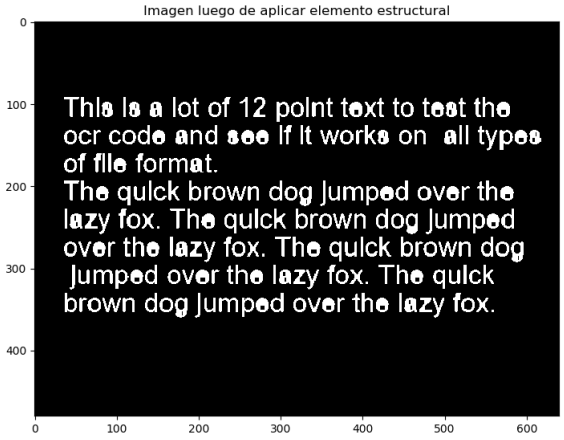

Figura 7: Imagen binaria luego de aplicar el elemento estructural.

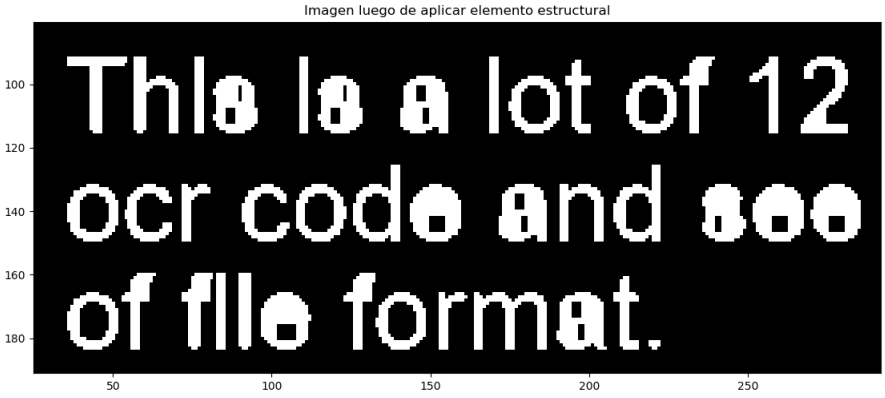

Figura 8: Imagen binaria luego de aplicar el elemento estructural (ampliada).

Como podemos observar, los píxeles de la imagen se juntan verticalmente, es decir, la 'i' se convierte en una 'l'. Esto soluciona el problema de contar caracteres erróneamente.

Ahora, aplicando el barrido e imprimiendo los recuadros sobre la imagen original obtenemos la siguiente imagen:

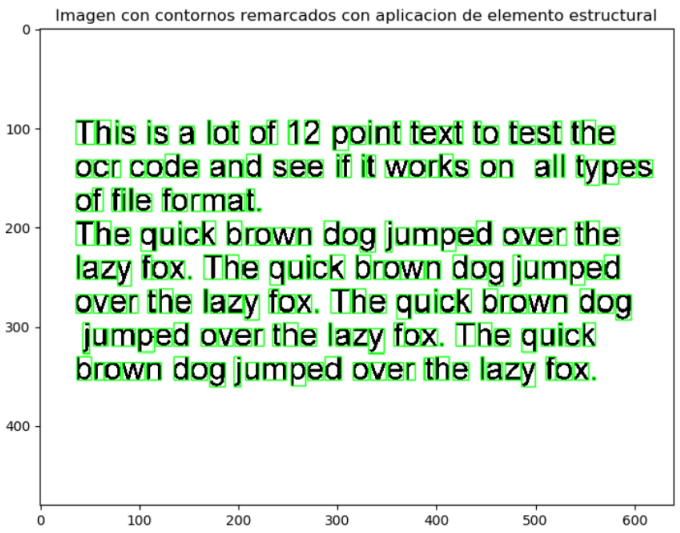

Figura 9: Imagen con contornos con elemento estructural aplicado.

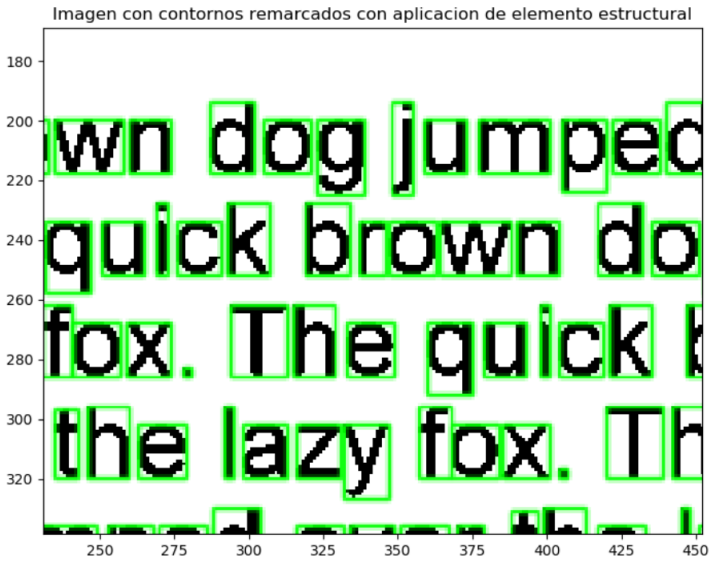

Figura 10: Imagen con contornos con elemento estructural aplicado (ampliada).

Como podemos apreciar, las letras 'i' y 'j' ahora poseen un solo recuadro en lugar de dos, es decir, ahora se contaran correctamente los caracteres. Este problema aplica para todos los caracteres que posean una parte separada, como por ejemplo '?', '!', etc.

3.6. Resultados

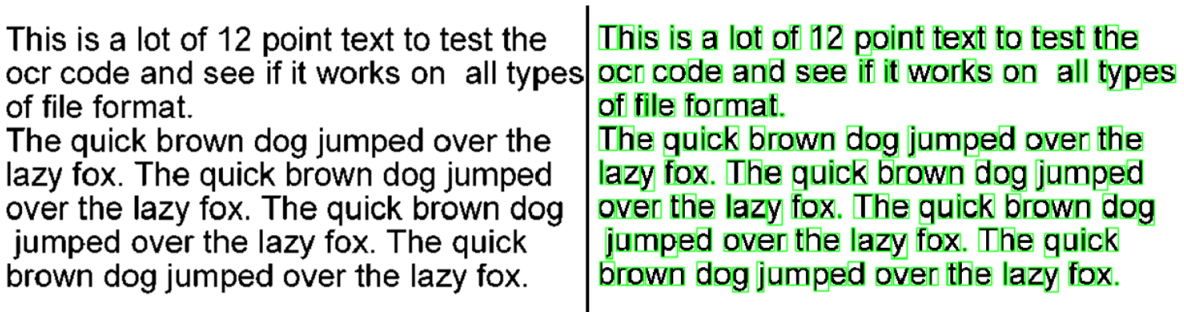

Figura 11: Resultados.

En la figura anterior podemos apreciar como el algoritmo rodea con un rectángulo verde cada caracter de una manera correcta, sin problemas con los caracteres que tienen píxeles separados. Para contar el numero de caracteres, simplemente tomamos la longitud del arreglo retornado en la función cv2.findContours(), que contiene el numero de contornos detectados en la imagen.

4. Optical Character Recognition (OCR)

Esta etapa es adicional a lo solicitado para la evaluación por lo que se coloco como material complementario. OCR es un proceso que consiste en la digitalización de textos a partir del reconocimiento de caracteres utilizando un algoritmo de machine learning que ya esta entrenado para determinar que segmentos de la imagen son caracteres o no. Es importante añadir que la imagen que se va a utilizar debe estar filtrada y binaria, por lo que se utilizaran los métodos antes explicados para generar esta imagen.

4.1. Instalación

Es necesario añadir dos librerías, así como también descargar e instalar la base de datos con el algoritmo de reconocimiento de caracteres implementado. Para esto se realizan los siguientes pasos:

Primero ingresamos al siguiente enlace `https://digi.bib.uni-mannheim.de/tesseract/tesseract-ocr-setup-3.05.01.exe` para descargar la versión 3.5 del algoritmo de reconocimiento de caracteres ya entrenado. Seguidamente se procede a ejecutar el instalador y luego de terminar es importante añadir la dirección de la carpeta donde se instalo en los PATH del sistema.
Luego añadimos las librerías necesarias a nuestro environment de Anaconda, para esto se ingresa por el terminal (en el caso de linux) o en el Anaconda Prompt (en el caso de Windows) los siguientes comandos:

```
> conda install pytesseract -n (nombre del emviroment)
```

```
> conda install pillow -n (nombre del emviroment)
```

4.2. Resultados

Para utilizar esta librería procedemos a ejecutar el método $pytesseract.image\_to\_string(image)$. Cabe destacar que image es una imagen que paso un filtrado de ruido y que se convirtió en una imagen binaria. El métodos antes mencionado retorna una cadena de caracteres con todo el contendido identificado, adicionalmente se puede seleccionar el lenguaje en que esta el texto incluyendo el parámetro lan='siglas del idioma'. Los resultados obtenidos tomando la imagen mostrada en la figura 3 son los siguientes:

```
El texto es el siguiente:
This is a lot of 12 point text to test the
cor code and see if it works on all types
of file format.

The quick brown dog jumped over the
lazy fox. The quick brown dog jumped
over the lazy fox. The quick brown dog
jumped over the lazy fox. The quick
brown dog jumped over the lazy fox.

Cantidad de caracteres:
225
```

Figura 12: Resultado del algoritmo OCR.