Universidad de Chile

Departamento de Computación

CC5508

Reconocimiento de caracteres usando binarización y componentes conexos

Profesor:
José M. Saavedra

Ayudante:
Francisco Clavero H.

Integrante:
Gabriela Mendoza M.

Rut:
19.566.930-9

Resumen

En esta entrega se reconocen los caracteres de modelos de imágenes que contienen una secuencia de números, puntos y guiones que componen un RUT. Esto, con el objetivo de profundizar los métodos de reconocimiento de caracteres vistos en cátedra, y analizar los pros y contras de cada algoritmo. Para lograr una identificación correcta de cada componente, se procede en cuatro pasos secuenciales:

1. Dado una imagen en 3 canales, se convierte a escala de grises.

2. La imagen en grises es binarizada. Se exploran tanto el algoritmo Otsu, como el Adaptativo para obtener el umbral o *threshold*.

3. Se extraen las componentes conexas de la imagen binaria, cada componente es representada por un identificador, una lista de puntos, una lista de puntos en el borde y un rectángulo mínimo que la contiene.

4. Por último se define y desarrolla una estrategia para caracterizar y reconocer cada símbolo, usando el borde de su componente conexa.

Introducción

En el marco del curso CC5508, Procesamiento y análisis de imágenes , se desarrollan algoritmos capaces de reconocer caracteres pertenecientes a una imagen RUT. Esto, con el objetivo de familiarizarse con los algoritmos de filtrado, intervención y caracterización de componentes en imágenes. Para esto, se cuenta con un conjunto de imágenes de muestra, e imágenes de modelo, sobre las cuales se trabaja.

La manipulación de imágenes es fundamental para todas las áreas que deben trabajar con material digitalizado o representaciones matriciales de fenómenos. Esto se extiende desde las áreas de investigación, médica, matemática y física, hasta las áreas de la industria, como la bancarias. También existen otros campo, más diversos, que requieren reconocimiento de imágenes, como la policía de investigaciones e incluso la cyber-seguridad. Es necesario lograr comprender los pasos fundamentales del preprocesamiento de imágenes, ya que , más allá de los algoritmos de aprendizaje de máquinas, o métodos estocásticos, un buen manejo de la entrada es fundamental para un reconocimiento exitoso.

Además de familiarizarse con estos métodos de pre-procesamiento, al final de este trabajo el estudiante logra manejar las librerías scipy ,sckimage y matplotlib para python. El manejo de estas herramientas será fundamental para esta entrega y trabajos futuros.

Desarrollo

Diseño del Programa

El programa diseñado se encuentra orientado a objetos, y cuenta con varias clases y subclases. Esto, con el objetivo de utilizar buenas prácticas de programación y reutilizar código.

- Se crea una clase GreyImage(Object) que implementa los métodos para la binarización de una imagen gris, y la obtención de las componentes conexas. Grey image es un composite de varias componentes conexas (clase Component), es decir contiene una lista de éstas.
- la clase Image extiende a GreyImage, Image recibe como parámetro la dirección (path) de una imagen rgb y el algoritmo (Otsu / adaptative) para la binarización. Además implementa un método que transforma una imagen en 3 canales, a una matriz de 1 canal.
- la clase Component cuenta con los atributos boundary, lpoints y boundingbox.
- Por último, el programa principal, main.py se encarga de establecer la interacción con el usuario.

Implementación del Programa

Algoritmo de detección de componentes

La clase Image cuenta con un arreglo de componentes conexas, donde cada componente es representada por un identificador, una lista de puntos, lista de bordes y un rectángulo mínimo que le cubre. En particular la lista de puntos se define como sigue: $l_\zeta = \{(i,j)|(i,j) \in \zeta\}$.

Para detectar la lista de puntos se siguen algoritmos de detección de componentes basados en el recorrido de pixeles vecinos, ocupando las estrategias *Depth First Search* y *Breadth First Search*.

En primer lugar se recorre la imagen en profundidad, y se revisa si el punto actual ya ha sido visitado antes en el grafo, de ser así, se continua en la siguiente iteración. De lo contrario el punto actual se marca como visitado y se verifica si es un punto negro (figura) o blanco (background). Al encontrar un punto negro se crea una nueva componente conexa, y se adjunta a la lista de componentes conexas de la clase GreyImage (lineas 6-8), además se incorpora el punto visitado a la lista de puntos de la componente (línea 9). Luego se procede a realizar una búsqueda en anchura sobre el punto encontrado y con la componente creada.

```
1     def dfs(self):
2         for i in range(self.rows):
3             for j in range(self.cols):
4                 if (i, j) in self.V: continue
5                 self.mark_visited((i, j))
6                 if (self.binimage[i][j] == 0):
7                     comp = Component(self.id)
8                     self.components.append(comp)
9                     comp.append_point((i, j))
10                    self.id += 1
11                    self.bfs_iterative(comp, i, j)
```

El algoritmo de búsqueda en anchura desarrollado recibe como parámetros un objeto Componente, y un punto i ,j. Se comienza con una cola FIFO vacía y de tamaña indefinido, y se almacena el punto actual, luego, mientras la cola no esté vacía se procede con la búsqueda (linea 4).

```
1         def bfs_iterative(self, comp, i, j):
2             myqueue = queue.Queue(maxsize=0)
3             myqueue.put((i, j))
4             while not myqueue.empty():
5                 x = myqueue.get()
6                 i = x[0]
7                 j = x[1]
8                 neighbours = [(i - 1, j), (i - 1, j + 1), (i, j + 1), (i + 1, j + 1),
9                               (i + 1, j), (i + 1, j - 1), (i, j - 1), (i - 1, j - 1)]
10                for x in neighbours:
11                    if self.in_range(x) and x not in self.V:
12                        if self.binimage[x[0]][x[1]] == 0:
13                            self.mark_visited(x)
14                            comp.append_point(x)
15                            myqueue.put(x)
```

En cada ciclo se recorre cada uno de los 8-vecinos del elemento primero en la cola, si este no ha sido visitado y está dentro del rango de la imagen se verifica que el punto pertenezca a la figura (línea 12). De ser así, el punto se marca como visitado y se añade a la lista de puntos de la componente, luego, se encola para que sus vecinos también sean visitados.

Algoritmo de detección de bordes

Cada componente conexa debe contar con una lista $lboundary_\zeta = \{(i,j)|(i,j) \in \Gamma\zeta\}$, donde $\Gamma(\cdot)$ es la función de extracción de bordes de un componente conexo, vista en clases.

En cada paso del algoritmo se toman los puntos cartesianos p sobre el borde, y q ,4-vecino de p, fuera del borde de la componente conexa. Para obtener el primer par p_0, q_0, se barren la imagen de forma left-rigth y top-bottom. También se prueba buscando el punto con menor distancia euclidiana al origen (borde superior izquierdo) de la grilla, obteniendo así p_0 y $q_0 = (p0[0], p0[1] - 1)$. Ambos métodos arrojan resultados similares, pero no idénticos, por lo que se opta por usar el primero.

Se tiene una función que entrega el siguiente 8 vecino de p, en sentido del reloj partiendo en q:

```
1     def next_neighbour(self, p, q):
2         i = p[0]
3         j = p[1]
4         n = [(i, j - 1), (i - 1, j - 1), (i - 1, j), (i - 1, j + 1),
5              (i, j + 1), (i + 1, j + 1), (i + 1, j), (i + 1, j - 1)]
6         indx = n.index(q)
7         if indx == len(n) - 1:
8             return n[0]
9         else:
10            return n[indx + 1]
```

Luego, mientras el siguiente a q no sea p_0, se recorren los 8-vecinos de p partiendo en q. En cada iteración se actualiza el valor de q a q_{sig}, y se mantiene una referencia al q previo q_{prev}. Si q_{sig} es de valor 0 (color negro), se actualiza p a q_{sig} y $q = q_{prev}$, con este método se asegura que $q = q_{prev}$ es 4-vecino de $p = q_{sig}$. De lo contrario, si q_{sig} es de valor 1, se continua en la siguiente iteración. Como se muestra en el siguiente algoritmo:

```
1         while (self.next_neighbour(p, q) != p0):
2             q_p = q
3             q = self.next_neighbour(p, q_p)
4             if (not self.in_range(q,image) or not image[q[0]][q[1]]):
5                 p = q
6                 q = q_p
7                 self.boundary.append(p)
```

Además, se debe tener en cuenta el caso de que el siguiente vecino esté fuera del rango matricial de la imagen. En este caso, cualquier punto fuera de rango, se considerará de color blanco (línea 4). Otra alternativa es agregarle un marco de "background" a la imagen.

Algoritmo de reconocimiento de símbolos

Algoritmo 1

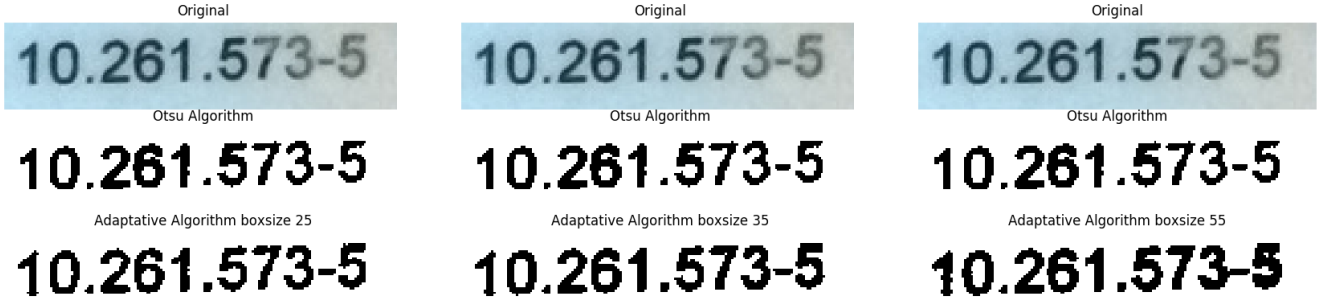
Para el reconocimiento de símbolos se procede utilizando un algoritmo que utiliza la descripción del borde de cada componente. Para lograr realizar una comparación de cada carácter con los modelos, primero se preprocesan los modelos y se obtiene el *feature vector* de cada uno. Para obtener este vector se realiza un histograma de direcciones, tomando los puntos vecinos del borde. Por cada par de puntos en el borde de la componente, se analiza en qué dirección está el segundo punto respecto al primero, se consideran en total 8 direcciones. Además se divide el bounding box en mitad superior e inferior, para obtener más precisión, y un vector final de largo 16. Luego de tener el histograma de direcciones se normaliza el vector con una funcón de numpy.

Teniendo el *feature vector* de cada modelo almacenado en un atributo de la clase Componente, se puede proceder en comparar cada carácter del RUT por todos los modelos. El parámetro escogido para describir la semejanza entre dos vectores es la distancia euclidiana, ya que considera la orientación espacial de cada componente. De esta comparación se escogen los mejores 10 resultados, y aquél con más repeticiones dentro de esta lista es escogido como el carácter reconocido.

Algoritmo 2

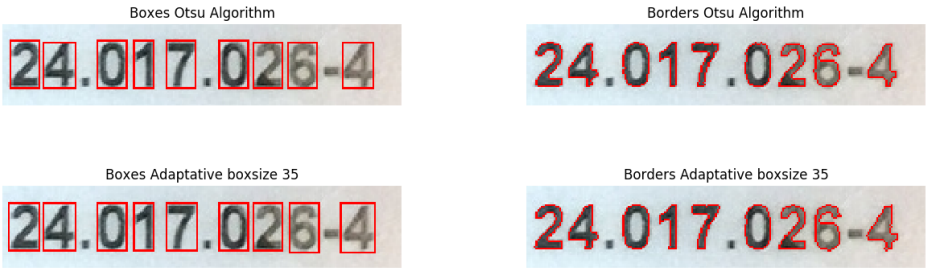
Se explora otra alternativa de algoritmo, utilizando el borde y el background de las componentes. Por cada elemento del background se "disparan 4 rayos" en las direcciones norte , sur este oeste. Cada punto del background se describe con un número de 4 bits 1 por cada dirección, si el rayo encuentra en su camino un punto del borde el bit correspondiente a la dirección del rayo toma el valor 1, de lo contrario, vale 0. Como cada punto cuenta con 4 bits característicos, en total hay $2^4 = 16$ combinaciones posibles. Luego, se realiza un histograma de las combinaciones, y se normaliza. Con este vector se procede a seleccionar el mejor modelo para la componente, con el mismo torneo descrito para el algoritmo 1.

Resultados



Al comparar el algoritmo Otsu, con el Adaptativo, en algunas imágenes se descarta que haya diferencia significativa. Sin embargo, en el caso de que la imagen cuente con una ilumninación no homogénea, el algoritmo Adaptativo genera una salida binaria con menos ruido y bordes más definidos que Otsu. Además el threshold para secciones de menor contraste, fue más sensibles en el algoritmo Adaptativo, por lo que se consideraban más puntos dentro de las componentes conexas, que en Otsu.

En el ejemplo anterior se observa que en el último tercio de la imagen original existe una diferencia significativa en la iluminación. En esta sección de la imagen el algoritmo Adaptativo logra detectar más puntos negros que Otsu. sin embargo, al disminuir el tamaño de la caja de barrido, el algoritmo arroja resultados más similares a Otsu. Por otro lado, al aumentar el *boxsize* de 35 a 55, el algoritmo sobrestima el threshold, generando figuras con bordes poco finos.



En las imágenes anteriores se aprecia el "bounding box" de cada RUT (imagen izquierda) y el borde detectado (imagen derecha). Se puede distinguir que el bounding box para el algoritmo adaptativo es más ancho que para Otsu, cuyas componentes están formadas por menos puntos. Además en la detección de bordes, los bordes de la imagen a la cual se le aplicó el algoritmo adaptativo son más suaves, sin embargo, debido al tamaño reducido de la imagen, esta diferencia no es considerable.

Reconocimiento: algoritmo 1 (histograma bordes)

:

Imagen	Original	Otsu	Adaptativo	Acierto1	Acierto2
rut1	15.407.110-5	15.054.115-5	13.6k711-k3	55,5%	44,4%
rut2	10.261.573-5	55.255.573-3	1k.551.328-8	44,4%	22.2%
rut3	24.017.026-4	06.115.125-2	26.k17.528-0	22.2%	44.4%
rut4	24.017.026-4	02.517.505-4	26.k12.k58-6	33,3%	22,2%
rut5	10.261.573-5	85.558.375-5	6k.586.328-2	22,2%	0%
rut6	10.261.573-5	15.051.575-5	1k.251.325-3	55,5%	33,3%
rut7	24.017.026-4	06.117.505-2	20.k12.528-6	22,2%	33,3%
rut9	10.457.232-4	11.257.232-2	15.637.282-8	66,6%	44,4%
rut10	15.068.718-7	15.555.715-5	15.k8k.215-2	44,4%	33,3%

* para obtener estos resultados se utilizaron tanto modelos nuevos cómo los originales.

En la tabla anterior, se muestra los resultados obtenidos utilizando el algoritmo de comparación de vectores característicos de las direcciones del borde de la componente. Se aprecia que la media para el algoritmo Otsu se encuentra alrededor del 44% de acierto. Es decir, reconoce, en promedio 4 de 9 caracteres. En cambio, los resultados para el algoritmo adaptativo (boxsize 35) tienen una media de acierto alrededor del 33%. Lo cual, es un comportamiento no esperado, ya que el algoritmo adaptativo es más sensible que Otsu.

Esta diferencia puede tener relación con el boxsize de la binarización adaptativa, o simplemente, se puede deber a una mala implementación del algoritmo. En los resultados del análisis anterior se observa cómo el boxsize afecta en la forma de la figura binarizada. Un boxsize demasiado grande puede generar caracteres indistinguibles.

Otro resultado importante es que para imágenes diferentes, pero que contienen el mismo RUT, se obtuvo resultados diferentes. Por ejemplo, en los RUT 2, 5 y 6, con el algoritmo Otsu, se obtuvo un 44, 22 y 55 % de acierto, respectivamente. En el ejemplo mencionado, se aprecia que las imágenes con mayor contraste logran un mejor resultado, la imagen con mayor contraste e iluminación es el RUT 6 que obtiene un 55% de acierto. Mientras que la imagen más oscura es el RUT 5 que obtiene tan sólo un 22% de acierto.

Reconocimiento: algoritmo 2 (rayos de direcciones del background)

:

Imagen	Original	Otsu	Adaptativo	Acierto1	Acierto2	Mejor
rut1	15.407.110-5	15.457.115-5	12.487.118-2	77,7%	55,5%	77,7%
rut2	10.261.573-5	15.551.573-5	18.281.273-2	66,6%	55.5%	66,6%
rut3	24.017.026-4	31.517.555-4	24.817.858-4	33.3%	55.5%	55.5%
rut4	24.017.026-4	34.517.554-1	54.817.828-4	33,3%	55,5%	55.5%
rut5	10.261.573-5	15.551.573-5	18.281.273-2	66,6%	55,5 %	66.6%
rut6	10.261.573-5	15.351.573-5	18.281.573-2	66,6%	66,6%	66.6%
rut7	24.017.026-4	34.517.535-4	24.817.858-4	44,4%	55,5%	55.5%
rut9	10.457.232-4	15.457.535-4	18.457.235-4	66,6%	77,7%	77.7%
rut10	15.068.718-7	15.555.715-7	18.888.718-7	55,5%	66,6%	66.6%

* para obtener estos resultados se utilizaron tanto modelos nuevos cómo los originales.

El segundo algoritmo de reconocimiento desarrollado arroja mejores resultados que el primero, acertando alrededor de 55% para Otsu y 60% para el algoritmo adaptativo, además, considerando el mejor de los dos algoritmos se consigue aproximadamente 66,6 % de acierto. A diferencia del algoritmo 1, este método muestra que el algoritmo adaptativo tiene un mejor desempeño promedio que Otsu, lo que es un comportamiento esperado. Esto se debe a que la binarización adaptativa se ajusta mejor a las variaciones de luz y contraste.

El problema con el método desarrollado es que al tomar el background de una imagen también se toman los puntos blancos encerrados por el borde de la componente. Es por esta razón, que caracteres como el 8 y el 0 y el 6 son difíciles de reconocer por el algoritmo, ya que sus vectores característicos con muy similares. Por otro lado este método no tiene tan buen alcance con la orientación espacial de la componente, es por esto que confunde el número 2 con el 5. Para solucionar estos problemas se podría crear dos secciones, una superior y otra inferior de la imagen, de esta forma se tiene un vector característico con el doble del tamaño del vector original.
Una mejora posible para este método es buscar en las 8 direcciones vecinas de cada punto, por lo que se obtendría un vector de largo 2^8. Sin embargo se debe considerar un aumento considerable en el tiempo de ejecución. Además para mejoras futuras se podrá distinguir entre el background fuera del borde y el background encerrado por el borde.

Conclusiones

No existe una forma certera de reconocer caracteres, sin asumir condiciones iniciales. Para saber qué método tiene mejor desempeño debemos tener conocimiento de cómo es la iluminación, contraste, orientación y tamaño de la imagen original. Basado en esto se puede escoger un algoritmo de binarización que se ajuste mejor a la imagen. Por lo visto en el trabajo, para imágenes con poco contraste funciona mejor el método adaptativo, mientras que de lo contrario, cualquiera de los dos sirve.

Una vez binarizada la imagen, se utilizan métodos que logran distinguir patrones basados tan solo en componentes conexas y detección de bordes. Tras finalizar esta entrega, se familiariza con estos métodos, y además se logra analizar cómo influye la binarización Otsu/Adaptativo en el desempeño de estos métodos.

Mirando en retrospectiva el trabajo realizado, se destaca la dificultad de realizar un algoritmo capaz de detectar 10 dígitos y 1 letra con gran precisión. La dificultad reside en la utilización de un algoritmo que compara los histograma de direcciones del borde de cada carácter (feature vector). Esta estrategia habría arrojado mejores resultados si hubiera sido posible entrenar el algoritmo antes de que el usuario interactuara con el programa. Un entrenamiento razonable habría sido utilizar las imágenes originales, un número de ellas que cubriera la cantidad de caracteres, y crear testings para los resultados del reconocimiento. Sin embargo, este tipo de aprendizaje, implica , aunque sea implícitamente utilizar las imágenes originales como modelos.

Por último, el algoritmo de disparar rayos obtuvo mejores resultados, ya que depende menos de las condiciones iniciales del input, que el algoritmo mencionado anteriormente. Además, este método deja la posibilidad de ser optimizado y mejorado para entregas futuras.