Tarea de Investigación No. 5
RECONOCEDOR ESTADÍSTICO DE DÍGITOS ESCRITOS A MANO CON EL USO DE HISTOGRAMAS COMO VECTOR DE ENTRADA.

Abiel Porras Garro - 2020209597[1], Elías Castro Montero - 2020098930 [2], Fabián Rojas Arguedas - 2019380107 [3], Roy García Alvarado - 2020109283 [4],
[1 2 3 4] Ingeniería en Computación, Instituto Tecnológico de Costa Rica, San José, Costa Rica

I. DESARROLLO

A. Reconocedores

1) Histograma

Uno de los métodos utilizados para la creación de un reconocedor fue el de obtener los valores de la imagen mediante histogramas. Estos indican la frecuencia (generalmente mediante gráficos) de los datos de una información entrante, en este caso una imagen. El objetivo es identificar el promedio de la frecuencia de características que tiene cada número en particular y poder crear un reconocedor a partir de las similitudes que se encuentren con la frecuencia de características del número que se ingrese para identificar.

Para encontrar estas características primero se debe realizar un pre-procesado de la imagen, donde es importante que el valor de los píxeles pase por una binarización para aislar las características de la imagen. Lo siguiente es dividir la imagen en filas y columnas de 4 píxeles de largo. En estas filas y columnas que se forman se debe analizar los píxeles, los píxeles para obtener los que son distintos e interpretarlos como características. Cada fila/columna tiene sus propios datos de frecuencia de características. Esta información es la utilizada para mostrar en las gráficas como en el siguiente ejemplo, donde en la parte izquierda se pueden visualizar las columnas pertenecientes a las frecuencias obtenidas en el eje Y, y en la parte de la derecha, las frecuencias obtenidas en el eje X:

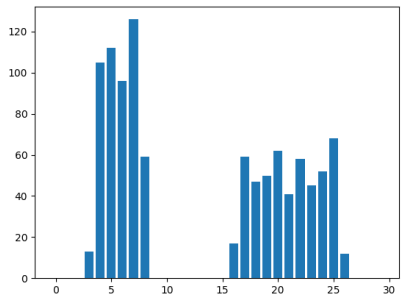

Fig. 1: Histograma del número 8

Se debe efectuar este proceso con todos los especímenes de cada número. Una vez obtenidos todos los histogramas, se debe llevar a cabo un promedio de todos los valores obtenidos para conseguir su media y luego obtener la desviación estándar. Con estos valores finalmente obtendríamos un modelo estadístico en el cual basarnos para hacer el reconocimiento.

Para proceder el reconocimiento se debe ingresar la imagen que se desea reconocer, a esta se le hace el pre-procesamiento y luego se obtiene su histograma. Ahora se deben seguir varios pasos para filtrar los posibles números a los que puede pertenecer el número entrante:

- Paso 1: Se pasa por el modelo de cada número para obtener su cantidad de características. El histograma de la imagen que está reconociendo debe tener por lo menos un 70% de la cantidad total de características para evaluar que posiblemente sea ese número. En el caso de que esto no se cumpla con ningún número, se concluye que la imagen no es un número. Si se da el caso en que se obtenga la misma cantidad de características que el modelo, entonces se guardan nada más los casos en que suceda esta situación.
- Paso 2: Ahora se debe filtrar los posibles casos hasta poder obtener uno solo. Para lograr esto, se juega con los márgenes de aceptación que tienen la misión de ir disminuyendo la desviación estándar poco a poco con cada iteración. Con esto poco a poco se van desechando casos que no calcen tan bien con el histograma del número que se está analizando hasta que solo quede una opción posible. Esta opción restante es la que se concluye que es el número que se está intentando reconocer.

2) Momentos Invariantes Hu

Este algoritmo es muy útil para reconocer imágenes con formas 2D con diferentes mapeos como lo son: traslaciones, orientaciones, rotaciones, escalados y reflexiones. Por ejemplo, algo muy básico que puede hacer este algoritmo es reconocer un cuadrado en cualquier posición que se encuentre.

¿Pero como funciona este algoritmo? Según [1] "Hu Moments (or rather Hu moment invariants) are a set of 7 numbers calculated using central moments that are invariant to image transformations. The first 6 moments have been proved to be invariant to translation, scale, and rotation, and reflection. While the 7th moment's sign changes for image reflection".

La explicacion anterior nos da una idea de manera general de como es que funciona el algoritmo pero aun faltan cosas de aclarar.

Los momentos de los que habla [1] son los ”momentos centrales”, los cuales no tienen mucho poder discriminatorio para representar formas, ni poseen propiedades invariantes. La formula general de los ”momentos centrales” la podemos apreciar en la figura2

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I(x, y)$$

Fig. 2: Fórmula para los momentos centrales

La formula de los momentos centrales puede llegar a ser muy útil dependiendo de como sea normalizada, aquí es donde entra en juego el algoritmo de los momentos invariantes Hu, este toma estos momentos relativos y crea 7 momentos separados que son adecuados para la discriminación de formas, estas 7 formulas la podemos apreciar en la figura 3.

$$M_1 = (\mu_{20} + \mu_{02})$$

$$M_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2)$$

$$M_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{30})^2$$

$$M_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2$$

$$M_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})((\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2) + (3\mu_{21} - \mu_{03})(\mu_{21} + \mu_{03})(3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2)$$

$$M_6 = (\mu_{20} - \mu_{02})((\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2) + 4\mu_{11}(\mu_{30} + 3\mu_{12})(\mu_{21} + \mu_{03})$$

$$M_7 = (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})((\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2) - (\mu_{30} - 3\mu_{12})(\mu_{21} + \mu_{03})(3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2)$$

Fig. 3: Fórmulas de los momentos invariantes Hu

Implementar este algoritmo en Python es relativamente sencillo ya que las librería de OpenCV ya trae las funciones necesarios para implementar este algoritmo, los pasos para obtener los momentos invariantes Hu son los siguientes:

$$H_i = -\text{sign}(h_i) \log |h_i|$$

Fig. 4: Fórmula para obtener los momentos invariantes de Hu en el mismo rango

- Binarizar la imagen, se puede lograr con la función de OpenCV ”threshold()”.
- Obtener los momentos centrales de la imagen binarizada con la función de OpenCV ”moments()”.
- Obtener los momentos Hu con los momentos centrales, utilizando la función de OpenCV ”HuMoments()”.
- Debido a que los números obtenidos de los momentos invariantes de Hu tiene un rango muy amplio, se debe utilizar la formula que se aprecia en la figura 4 para traer los números al mismo rango.

II. Comparación entre reconocedores

A. Tabulación de resultados

Con respecto a los resultados obtenidos con ambos reconocedores, la primera tabla muestra la tasa de aciertos del primer reconocedor que utiliza histogramas, el cual da como resultado un total de aciertos de 1628 y un total de desaciertos de 352, es decir, el porcentaje de aciertos totales del reconocedor es de $\mathbf{82.22}\%$.

Dígito	Aciertos	Desaciertos	Porcentaje de Exactitud
0	173	25	87.37%
1	146	52	73.74%
2	158	40	79.8%
3	133	65	67.17%
4	173	25	87.37%
5	182	16	91.92%
6	163	35	82.83%
7	178	20	89.9%
8	178	20	89.9%
9	144	54	72.73%

Por otro lado, la segunda tabla muestra la tasa de aciertos del segundo reconocedor que utiliza momentos invariantes Hu, el cual da como resultado un total de aciertos de 730 y un total de desaciertos de 1249, para un porcentaje de aciertos del $\mathbf{36.89}\%$.

Dígito	Aciertos	Desaciertos	Porcentaje de Exactitud
0	167	31	84.34%
1	139	58	70.20%
2	46—	152	23.23%
3	43	155	21.71%
4	79	119	39.89%
5	2	196	0.01%
6	72	126	36.36%
7	60	138	30.30%
8	101	97	51.01%
9	21	177	10.60%

B. Análisis de los resultados

Como es posible observar el reconocedor que utiliza histogramas como entrada, es mucho más acertado. Además, se debe tomar en cuenta que el preprocesamiento fue el mismo para las imágenes que utilizan ambos reconocedores, sin embargo, debido a la naturaleza de los momentos invariantes Hu, para el segundo reconocedor se eliminó la característica encargada de centrar el dígito en la imagen.

Se realizaron diversas variaciones del reconocedor (y su preprocesamiento) que utiliza momentos Hu con el objetivo de incrementar su tasa de aciertos, que se describen a continuación:

- Se redujo la dilatación de la imagen, pasando de 3 iteraciones de dilatación a 2, la tasa de aciertos se mantuvo en 36.89%.
- Se aplicó el algoritmo Canny Edge detection para utilizar los bordes de los números en vez del dibujo del número como tal, la tasa de aciertos fue de 36.89%.
- Se dejó de utilizar el algoritmo Canny Edge detection, la tasa de aciertos fue de 32.98%.
- Se probó intentar quitar la redimensión de las imágenes en una prueba, y en otra prueba se dejó de obtener el negativo del número, es decir, se mantuvo el dígito escrito en color negro y el fondo de la imagen de color blanco, sin embargo, en ambos casos no se obtuvo una mejora en los resultados, incluso al utilizarse especímenes de distinto tamaño, existen mayores conflictos y mayor

trabajo adicional, ya que el código debe ser adecuado para trabajar con tamaños diversos y desconocidos.

Dentro de las posibles explicaciones que pueden existir para esta gran diferencia está que los dígitos tienen un vector de momentos Hu muy parecido entre sí, ya que al ser una de las principales diferencias los ángulos en los que se escriben los dígitos y este dato perderse por la invariabilidad que manejan estos momentos, se pierde información importante que podría ser útil para distinguir números, por ejemplo, la figura 5 representa el dígito 1, sin embargo, al perderse la variabilidad de la dirección y ángulo del dígito, este podría ser reconocido como un 7, sin embargo, no se cuenta con esta información cuando se utilizan momentos Hu por lo que la tasa de aciertos podría verse afectada.

Fig. 5: Especímen del dígito 1, que podría ser confundido con un 7

Finalmente, vale la pena destacar los grandes resultados obtenidos al utilizar histogramas como insumo para el reconocedor de dígitos, a pesar de su simplicidad, esta técnica es más efectiva que los momentos invariantes Hu, sin embargo, requieren de un mayor preproceso, ya que los valores del histograma no son invariantes y son sensibles a cambios en ángulos y direcciones, lo cual en este trabajo fue útil, pero en otros puede que no sea lo ideal.

C. Conclusión de la comparación entre ambos reconocedores

- Una gran ventaja de utilizar HuMoments es que la orientación y las dimensiones del número que se desea reconocer pueden ser bastante libres, ya que igualmente va a obtener los mismos resultados debido a que los bordes siempre van a ser los mismos. Esto es de gran ayuda cuando se requiera hacer un reconocedor dónde las imágenes que analice no tengan la mejor calidad, aún en estas condiciones se presta para hacer un buen trabajo.
- El método HuMoments tiene la desventaja de que es más difícil de implementar que otros métodos. Esto debido a que su funcionamiento interno es más complejo que en otros algoritmos y esto tiende a hacer que sus usuarios no entiendan bien como utilizarlo. Por otro lado, se tiene que tener clara la forma en que se desea integrar al programa para que durante el desarrollo se tenga esto en mente y el programa aproveche al máximo los resultados que se pueden obtener de este método, al mismo modo se debe realizar un buen manejo de especímenes.
- Debido a que los dígitos escritos a mano que se utilizan para probar ambos algoritmos son dígitos sin ningún tipo de rotación, escalado, espejo o cualquier otro tipo de mapeo; el algoritmo del histograma es mas útil en este caso en especifico, ya que solo verifica que el numero de píxeles en los ejes ”X” y ”Y”.
- En el caso del algoritmo de momentos invariantes Hu, muestra resultados muy diferentes entre muestras del mismo dígito, lo que quiere decir que para obtener resultados mas exactos con este algoritmo es necesario que se ”entrene” y se pruebe con dígitos muy similares entre si. En caso contrario, con el algoritmo del histograma, a pesar de que los especímenes para entrenar el reconocedor no son muy similares entre si, genera un buen histograma que da muy buenos resultados al momento de reconocer los dígitos escritos a mano.

La técnicas de preprocesamiento que dieron los mejores resultados en el reconocedor que utiliza momentos Hu, fueron la redimensión de los especímenes a un tamaño fijo y la extracción de bordes de los dígitos utilizando Canny Edge detection.

III. Conclusiones

- Los momentos Hu logran dar aun mas características que un histograma, ya que este sin mucho procesamiento logra dar datos sobre que tan redondeado es un elemento en una imagen sin importar su forma. Con el histograma seria posible, pero se necesitaría un análisis posterior de los datos de las ventanas de píxeles.
- Hay métodos que, aunque a primera vista parezcan que parecen adecuarse a las necesidades que tenemos, como es el caso del HuMoments, puede darse la casualidad de que no sean tan efectivos debido a la manera en que se está abordando el problema. Sin embargo, como demostró el método mediante histogramas, siempre puede haber otra opción que se adecue mejor a nuestra situación. Por lo que siempre es bueno probar distintas alternativas para encontrar la que mejores resultados nos pueda dar.
- El uso de especímenes de buena calidad junto con un correcto pre-procesamiento son dos de las partes que mas importancia y que se deben tratar con mucho cuidado ya que afectan directamente el resultado de los reconocedores, un mal preprocesamiento que genere imágenes muy disparejas va a ocasionar que las muestras sean muy disparejas y utilizar imágenes de buena calidad facilita a lograr una separación clara de los distintos casos ´ de d´ıgitos.
- Ambos algoritmos utilizan un preprocesamiento muy similar ya que en ambos casos se deben binarizar las imágenes, sin embargo, la principal diferencia de estos algoritmos al momento de la práctica es que los momentos Hu además de diferenciar los dígitos escritos a manos, también es capaz de reconocerlos con diferentes mapeos o posiciones de un dígito, esto ultimo no lo hace el histograma.
- Una vez realizados los distintos algoritmos, podemos ver la importancia de escoger especímenes más estandarizados con el fin que las estadísticas sean mas cerradas y por lo tanto con una mejor selección. Aunque el poseer especímenes variados puede ayudar a detectar mas ejemplares, también provoca que un sistema sea

más propenso a equivocaciones al ampliar el margen de aceptación debido a la gran variedad.

REFERENCES

[1] S. Mallick, *Shape Matching using Hu Moments (C++/Python)*. LearnOpenCV, december 2018. [Online]. Available: https://learnopencv.com/shape-matching-using-hu-moments-c-python/