Proyecto de Desarrollo 1
IMPLEMENTACIÓN Y COMPARACIÓN DE DISTINTOS MÉTODOS PARA GENERAR UN RECONOCEDOR DE DÍGITOS ESCRITOS A MANO

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

I. INTRODUCCIÓN

Las redes neuronales son una técnica de inteligencia artificial que enseña a las computadoras a procesar datos de una manera que imita el cerebro humano. Es un proceso de aprendizaje automático llamado aprendizaje profundo que utiliza nodos o neuronas conectadas en una estructura jerárquica similar al cerebro humano [1].

El presente trabajo consiste en investigar, analizar y comparar diferentes métodos para realizar un reconocedor de dígitos numerales escritos a mano, así como también se pretende entender su funcionamiento con el fin de obtener el conocimiento suficiente para poder ser aplicado y adaptado en diferentes situaciones.

En especifico se muestran tres diferentes métodos reconocedores de dígitos escritos a mano, los cuales son: Histogramas, Momentos invariantes de Hu y Redes Neuronales. De los cuales, los primeros dos se tomaron de las tareas de investigación 4 y 5 realizadas para el curso de Introducción al Reconocimiento de Patrones en el segundo semestre del año 2022, el tercer método se basó en el código desarrollado por Patel y publicado en GitHub. Con los tres métodos previamente mencionados se realizará un estudio comparativo de manera sistematizada con el fin de determinar de forma cuantitativa la eficacia de cada uno de los modelos, para indicar por medio del estudio cual resulta ser el mejor.

A. Descripción del proyecto

Este proyecto consiste en un reconocedor de dígitos escritos a mano realizado mediante el método de redes neuronales. Las redes neuronales son un método de la inteligencia artificial donde las computadoras aprenden mediante el procesamiento de datos de manera muy cercana a como lo haría una mente humana, ya que se comporta como un modelo simple del funcionamiento del sistema nervioso. Las neuronas son las unidades de este método, generalmente organizadas por capas.

Las capas con las que cuenta la red neuronal de este proyecto son las siguientes:

1) *Flatten layer:* Capa encargada de aplanar cada especimen, es utilizada ya que las imágenes son matrices bidimensionales. Para crear la capa se utiliza la clase `Flatten` que recibe las dimensiones de las entradas, en nuestro caso 67x50 [2].

```
keras.layers.Flatten(input_shape=(67,
    50)).
```

2) *Dense layer:* Capa densa, esto significa que cada neurona de esta capa está conectada con todas las neuronas de las capa anterior. En nuestro caso, cada una de las 100 neuronas de esta capa densa está conectada con todos píxeles (neuronas) de la imagen que se recibe como entrada [2]. Además, se utiliza la función de activación `relu`, ya que aplica el **Rectificador Lineal Unitario** (ReLu) sobre las neuronas, para introducir la no linealidad en las redes neuronales, de forma que las neuronas que tengan valores negativos se apaguen/desactiven, debido a que su formula es $relu(x) = max(x, 0)$ [2]. La ventaja de emplear esta función de activación es que las neuronas pueden tener pesos más reducidos al limitar el rango de valores que pueden tomar.

```
keras.layers.Flatten(input_shape=(67,
    50)).
```

3) *Dense layer:* De nuevo, se emplea una capa densa con las mismas propiedades de conexión que la capa anterior, esta es la capa de salida que está compuesta por 10 neuronas, donde cada una representa uno de los número del 0 al 9 [2]. Además, en este caso se utiliza la función de activación sigmoide (**Sigmoid function**) cuya fórmula es $sigmoid(x) = \frac{1}{(1+exp(-x))}$. La principal ventaja de esta función es que su rango es de 0 a 1, lo cual en el caso de las capas de salida (*output layers*) es muy útil, ya que se puede interpretar como una probabilidad que puede ir de 0 a 1 [2]. Posteriormente se explicará como estas probabilidades son utilizadas por el modelo reconocedor para decidir a cual dígito corresponde una imagen.

```
keras.layers.Flatten(input_shape=(67,
    50)).
```

Estas tres capas conforman el modelo secuencial que

tendrá la red neuronal. Después de haber creado el modelo, se debe de configurar para su entrenamiento, esto se puede realizar mediante la función `Model.compile`, que recibe argumentos como el optimizador y la función de perdida (loss function), en nuestro caso utilizamos el optimizador que utiliza el algoritmo de Adam, un método de descenso de gradiente estocástico basado en estimaciones de momentos de primer y segundo orden, el cual ofrece un buen rendimiento en problemas con muchos información o parámetros [3][4]. Como función de perdida se emplea la función `sparse_categorical_crossentropy` que calcula la **Entropía cruzada categórica** (*categorical cross entropy*), ya que suele ser utilizada cuando la salida de la red neuronal representa probabilidades, lo cual es cierto, ya que la capa de salida utiliza la función de activación sigmoide [3].

Finalmente, se debe entrenar al modelo. Esto se puede llevar a cabo mediante varias iteraciones sobre un dataset. Por ejemplo, si quisiéramos realizar un entrenamiento de 5 iteraciones se debe utilizar la función `model.fit(X_train, y_train, epochs=5)` [5].

II. Diagrama de Bloques

En esta sección se explicará el diagrama de bloques en tres niveles diferentes empezando por el nivel más general y finalizando por el nivel mas específico. En los diferentes diagramas de bloque se observa como funciona un sistema reconocedor de dígitos numerales escritos a mano utilizando una red neuronal.

A. Tercer Nivel

El tercer y último nivel del diagrama de bloques se puede observar en la figura 1 en la que se observa de manera Especifica como fue implementado el sistema reconocedor de dígitos numerales escritos a mano utilizando una red neuronal. Cada uno de estos bloques se explica a continuación.

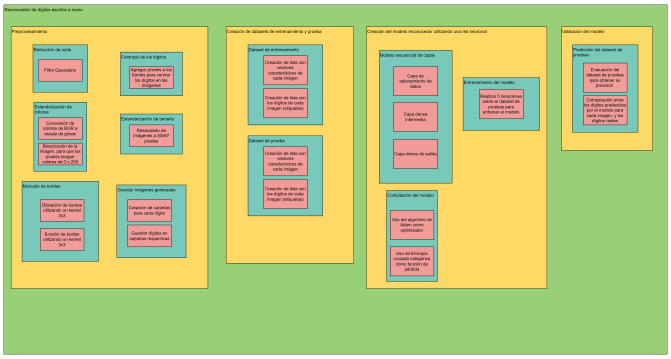

Fig. 1: Diagrama de bloques del reconocedor

Para el bloque Preprocesamiento:

- **Filtro gaussiano:** Se utiliza el filtro gaussiano con el fin de reducir el ruido en la imagen y suavizar los bordes. La biblioteca OpenCV trae una función para aplicar un filtro gausseano a la imagen de la siguiente manera:

```
cv2.GaussianBlur(img, (5, 5), 0)
```

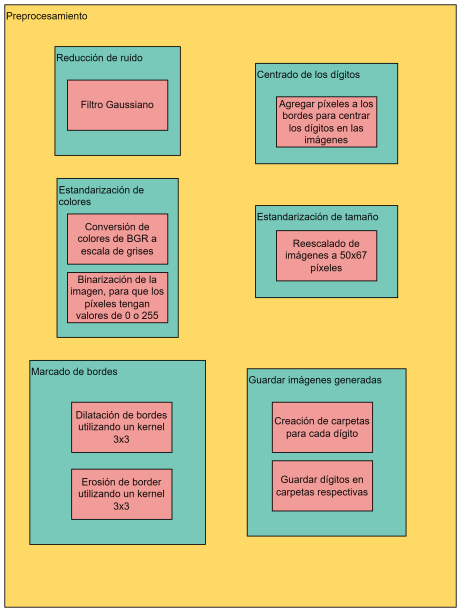

Fig. 2: Diagrama de bloques del bloque Preprocesamiento

- **Conversión de colores de BGR a escala de grises:** Con el fin de estandarizar los colores, se convierte a escala de grises de tal forma que cada píxel solo va a tener un vector que va de 0 (para blanco) a 255 (para negro). La biblioteca OpenCV trae una función para convertir las imágenes de BGR a escala de grises de la siguiente manera:

$$\text{cv2.cvtColor(img, cv2.COLOR}_{BGR2GRAY})$$

- **Binarización de la imagen:** Esto con el fin de diferenciar mas sencillamente los píxeles que son parte del dígito, en este paso también se convierten los píxeles negros a blancos y los píxeles blancos a negros, de tal forma que 0 va a representar las secciones que no tienen forman parte del dígito.
- **Se agregan píxeles a los bordes para centrar la imagen:** Se le agregan filas y columnas a cada uno de los lados de la imagen con un tamaño de 10 píxeles a partir de donde se encontró el primer píxel negro, de esta manera todas las imágenes quedan centradas por igual.
- **Reescalado de imágenes a 50x67 pixeles:** Después de todo el preprocesamiento, cambios y filtros que se le aplicaron a la imagen se procede a reescalar la imagen para que todas tengan el mismo tamaño y sean lo mas estandarizadas posible.
- **Dilatación y erosión de los bordes:** Esto con el fin de marcar mejor los bordes. La biblioteca OpenCV trae por defecto las funciones para dilatar y erosionar los bordes de una imagen, se realiza de la siguiente manera:

```
cv2.dilate(img, kernel, iterations=1)
cv2.erode(img, kernel, iterations=1)
```

Para el bloque Creación de datasets de entrenamiento y prueba:

- **Dataset de entrenamiento**: Se utiliza un 70% de los especímenes

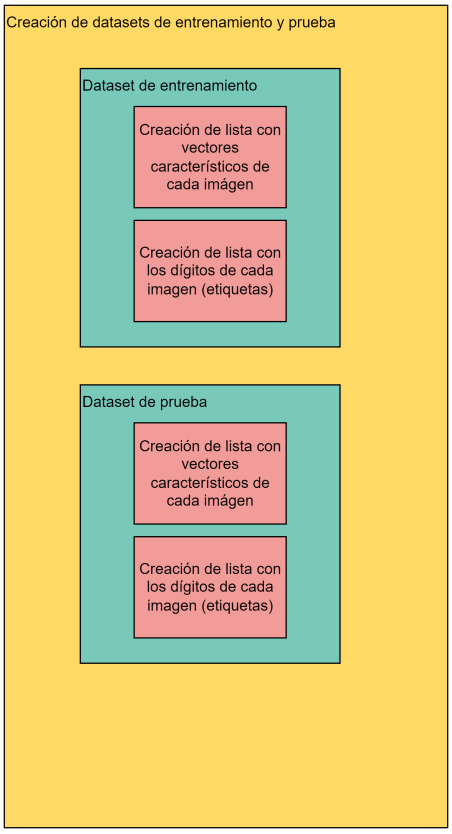

Fig. 3: Diagrama de bloques del bloque Creación de datasets de entrenamiento y prueba

  - ***Creación de la lista con vectores característicos de cada imagen:*** Se obtiene una lista con los píxeles de cada imagen.
  - ***Creacion de lista con los dígitos de cada imagen:*** Se obtiene una lista con el dígito al que corresponde cada imagen.
- **Dataset de prueba**: Se utiliza un 30% de los especímenes
  - ***Creación de la lista con vectores característicos de cada imagen:*** Se obtiene una lista con los píxeles de cada imagen.
  - ***Creacion de lista con los dígitos de cada imagen:*** Se obtiene una lista con el dígito al que corresponde cada imagen.

Para el bloque Creación del modelo reconocedor utilizando una red neuronal:

Las capas con las que cuenta la red neuronal se encuentran en un modelo secuencial de capas que contiene: Capa de aplanamiento de entrada, capa densa intermedia y capa densa de salida acorde con lo explicado en la Introducción.

El modelo se compila utilizando el algoritmo de Adam como optimizador y Entropía cruzada categórica para la función de pérdida.

Finalmente se entrena el modelo con 5 iteraciones sobre el dataset de entrenamiento.

Para el bloque Validación del modelo:

- **Evaluación del dataset de pruebas para obtener su precisión**: Se evalúa la precisión del modelo la función `model.evaluate(X_test, y_test)` que compara los resultados de la red neuronal aplicada al dataset de prueba con los dígitos reales para cada imagen.
- **Comparación entre las predicciones y los dígitos reales**: Se utiliza la función `model.predict(X_test)` para obtener una predicción para cada dígito del dataset de pruebas y se compara con el real. Esto se realiza para obtener la cantidad de aciertos y fallos de la red neuronal por

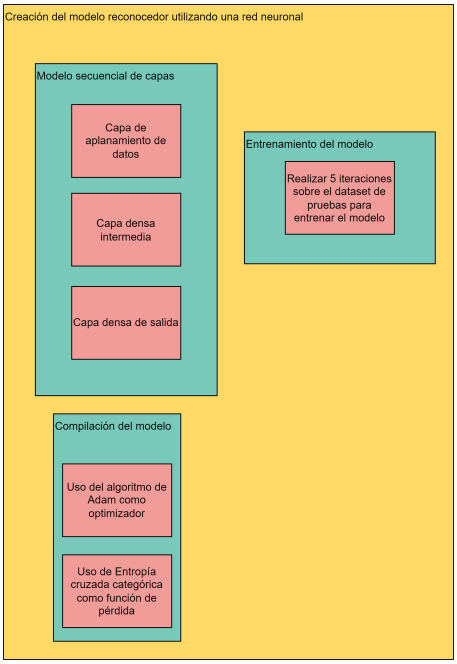

Fig. 4: Diagrama de bloques del bloque Creación del modelo reconocedor utilizando una red neuronal

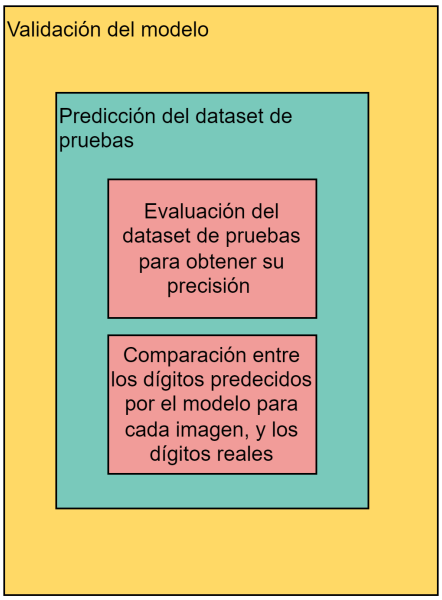

Fig. 5: Diagrama de bloques del bloque Validación del modelo

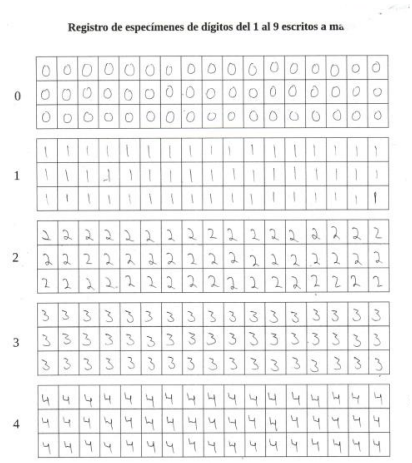

Fig. 6: Ejemplo de plantilla utilizada para obtener los dígitos numerales escritos a mano

cada dígito. Luego se imprime mediante una matriz de confusión estos valores.

III. Reconocedores anteriores

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:

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

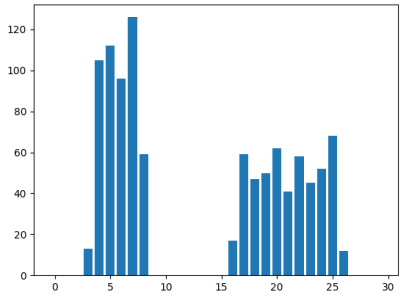

Fig. 7: Histograma del número 8

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 [6] ”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 [6] 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 figura8

La formula de los momentos centrales puede llegar a ser muy útil dependiendo de como sea normalizada, aquí es

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

Fig. 8: Fórmula para los momentos centrales

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 9.

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

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

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

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

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

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

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

Fig. 9: 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. 10: 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 10 para traer los números al mismo rango.

IV. Resultados y Análisis

A. Tabulación de resultados

Con respecto a los resultados obtenidos en los diferentes reconocedores de dígitos numerales escritos a mano, podemos observar que en la tabla I muestra la tasa de aciertos del reconocedor que utiliza histogramas, 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 aproximadamente de $\mathbf{82.22}\%$ y el porcentaje de desaciertos es de aproximadamente $\mathbf{17.78}\%$.

La tabla II 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, es decir, el porcentaje de aciertos totales del reconocedor es aproximadamente de $\mathbf{36.89}\%$ y el porcentaje de desaciertos es de aproximadamente $\mathbf{63.11}\%$.

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%

TABLE I: Tabla de resultados de reconocedor de dígitos numerales escritos a mano utilizando histogramas

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%

TABLE II: Tabla de resultados de reconocedor de dígitos numerales escritos a mano utilizando momentos invariantes de Hu

La tabla III muestra la tasa de aciertos del tercer reconocedor que utiliza redes neuronales, el cual da como resultado un total de aciertos de 1940 y un total de desaciertos de 40, es decir, el porcentaje de aciertos totales del reconocedor es aproximadamente de $\mathbf{97.98}\%$ y el porcentaje de desaciertos es de aproximadamente $\mathbf{2.02}\%$.

Dígito	Aciertos	Desaciertos	Porcentaje de Exactitud
0	194	4	97.98%
1	196	2	98.99%
2	192	6	96.97%
3	195	3	98.48%
4	194	4	97.98%
5	190	8	95.96%
6	197	1	99.49%
7	195	3	98.48%
8	194	4	97.98%
9	193	5	97.97%

TABLE III: Tabla de resultados de reconocedor de dígitos numerales escritos a mano utilizando redes neuronales

B. Análisis de los resultados

Como se puede observar el reconocedor que utiliza histogramas como entrada, es mucho más acertado que el reconocedor implementado con momentos invariantes de Hu, sin embargo, no se compara con los excelentes resultados del reconocedor implementado con redes neuronales. Es importante mencionar, que el preprocesamiento fue el mismo para las imágenes que utilizan los tres reconocedores, y además utilizan las mismas

imágenes para ser entrenados y probados, por lo que se puede decir que los tres reconocedores al momento de ser probados estuvieron en igualdad de condiciones; con excepción de que, debido a la naturaleza de los momentos invariantes Hu, 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.

Como se puede observar claramente, el reconocedor menos efectivo es el que utiliza momentos invariantes de Hu, 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 11 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. 11: Espécimen del dígito 1, que podría ser confundido con un 7

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.

Por ultimo, el reconocedor utilizando redes neuronales fue el que presentó los mejores resultados de los tres métodos por mucho, con resultados casi perfectos como se puede observar en el caso del dígito 6 que solo tuvo un desacierto de los 198 dígitos que analizó; este método utiliza exactamente el mismo preprocesamiento que el método que utiliza histogramas, y con la ayuda de las bibliotecas keras, numpy, entre otras que se pueden utilizar con python, este método se hace confuso de entender pero fácil de implementar, el nivel de dificultad de implementación se podría asemejar al método que utiliza histogramas, e incluso podría considerarse como más sencillo, ya que simplemente se debe de hacer uso de las funciones que ofrecen las bibliotecas.

V. Conclusiones

- Los métodos de momentos invariantes de Hu y el uso de histogramas es útil para casos muy específicos en el que los datos se puedan interpretar como vectores de dos dimensiones aun así no llegan a acercarse a los resultados del reconocedor implementado con redes neuronales, tomando en cuenta que las redes neuronales son más flexibles con respecto al tipo de dato y las características que se desean analizar.
- Con el uso de redes neuronales en cada capa de nodos se le puede especificar el tipo de dato o característica que va a abarcar dicha capa, por lo que se pueden tener múltiples características para diferenciar los especímenes entre si, esto no es posible con los otros dos métodos reconocedores implementados
- Como fue posible observar a la hora de desarrollar un reconocedor se debe de tomar en cuenta detalles como la naturaleza de los especímenes con los que se cuenta, por ejemplo, en el caso del reconocedor que utiliza momentos Hu, al no existir el factor de la variabilidad, se pueden perder detalles como los ángulos de rotación de los dígitos, por ende, para este caso, no nos fue de mucha utilidad, ya que eran necesarios para distinguir números que pueden ser escritos de forma parecida; sin embargo, para otros contextos, puede que la invariabilidad sea uno de los requerimientos más importantes, por ende, es importante entender con qué datos se cuenta para elegir las herramientas que más se adecúen al contexto presentado.
- Las redes neuronales han evolucionado al punto en que con unas cuantas llamadas a bibliotecas como Keras, se pueden obtener muy buenos resultados. Esta es una de las principales tendencias de los últimos años, y viendo los resultados que se obtuvieron cuando se utilizaron como reconocedor, se puede concluir de forma contundente, que realmente es una tecnología que vale la pena aprender, tanto por la facilidad que existe hoy en día para su uso, sino que también por su versatilidad, ya que es muy útil en áreas como medicina, procesamiento de imágenes, robots, economía, modelos estadísticos, simulaciones, en general, es aplicable a una cantidad inimaginable de casos de uso.

VI. Recomendaciones

- A la hora de utilizar tecnologías relacionadas a inteligencia artificial y machine learning, como redes neuronales, es muy fácil empezar a utilizar funciones que obtengan resultados favorecedores, sin embargo, esto agrega una capa de abstracción que podría afectar el conocimiento sobre el sistema del programador, ya que no se conoce con exactitud que herramientas o métodos se utilizan para implementar las funciones, lo cual complica la capacidad de realizar modificaciones o reparaciones en el código. Por ende, se sugiere visitar la documentación de las funcionalidades externas que se vayan a utilizar, para entender realmente como funcionan, con el objetivo de elegir aquellas que sean más convenientes de acuerdo a nuestras necesidades, como en este caso fueron las capas de la red neuronal y la función de pérdida y activación.
- Se sugiere emplear representaciones de resultados de forma más gráfica como pueden ser tablas que resuman los resultados obtenidos para cada una de las clases o etiquetas (dígitos, en nuestro caso), estas permiten mejor de forma visual el rendimiento obtenido en nuestros reconocedores, a la vez de que permite darnos cuenta de detalles que a simple vista podrían no ser observados si nos preocupase únicamente el % de aciertos final del sistema, por ejemplo, gracias a la tabulación de los resultados, notamos que cuando se utilizan histogramas hay dígitos que son reconocidos con un porcentaje de aciertos igual o mayor al 90%, mientras que hay otros son reconocidos con un porcentaje mucho menor de menos de un 70%.
- En situaciones en las que se deban desarrollar sistemas y ya se cuente con módulos, bibliotecas o herramientas existentes; crear un diagrama de bloques permite identificar aquellos bloques en donde pueden utilizarse, a la vez que se puede organizar el sistema en diversos bloques, donde existan aquellos existentes y otros que se deban desarrollar. Por ejemplo, en nuestro caso fue de utilidad, para modularizar el reconocedor de dígitos escritos a mano, ya que existían módulos que habían sido creados en los trabajos anteriores, como los de preprocesar las plantillas que contenían los dígitos; por lo que bastaba con identificar los bloques que debían desarrollarse, para iniciar con la implementación del sistema.

VII. Repositorio

Código GitHub

References

[1] "¿Qué es una red neuronal? Guía de IA y ML - AWS." [Online]. Available: https://aws.amazon.com/es/what-is/neural-network/
[2] f. given i=K., given=Keras, "Keras documentation: Keras layers API." [Online]. Available: https://keras.io/api/layers/
[3] ——, "Keras documentation: Losses." [Online]. Available: https://keras.io/api/losses/
[4] ——, "Keras documentation: Optimizers." [Online]. Available: https://keras.io/api/optimizers/
[5] ——, "Keras documentation: Model training APIs." [Online]. Available: https://keras.io/api/models/model_training_apis/
[6] S. Mallick, *Shape Matching using Hu Moments (C++/Python)*. LearnOpenCV, december 2018. [Online]. Available: https://learnopencv.com/shape-matching-using-hu-moments-c-python/