Tarea de Investigación No. 4
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. Tasa de aciertos

Para calcular la tasa de aciertos, primero se genera el modelo estadístico que contiene el valor promedio y varianza del histograma de cada uno de los diez dígitos utilizando el 70% de los especímenes (465 imágenes por cada dígito), posteriormente este modelo estadístico es utilizado como reconocedor, y se prueba su efectividad utilizando el 30% de los especímenes (198 imágenes por cada dígito), primero se envía la imagen al reconocedor y se compara el número que retorna el reconocedor (el dígito al que dice que corresponde la imagen) con el dígito que en realidad es.

A continuación, se muestra la tasa de aciertos de nuestro reconocedor para cada dígito utilizando 198 especímenes para cada dígito:

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%

Los resultados anteriores dan 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 $\mathbf{82.22}\%$.

Desde nuestro punto de vista es un resultado bastante aceptable, aunque tiene ciertos puntos de mejoras principalmente en dígitos como el 1, 3 y 9; pensamos que puede deberse a la variedad de formas en las que se pueden escribir estos dígitos, por ejemplo, la figura 1 muestra tres formas distintas de escribir un 1, en donde incluso la segunda imagen podría ser confundida con un 7. Por ende, el reconocedor actualmente es sensible a la forma en la que se escriban los números, por lo que como tarea pendiente queda mejorar este aspecto, tal vez recabando más información sobre el número. No obstante, también se debe destacar el hecho de que hay dígitos como el 5, 7, 8 los cuales el reconocedor es capaz de identificarlos con una efectividad alrededor al 90% lo que indica que está haciendo un muy buen uso de la información generada por el histograma, y su valor esperado y varianza.

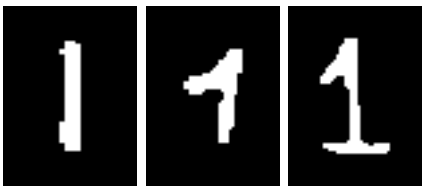

Fig. 1: Tres formas distintas de escribir el dígito 1

Al elegirse aleatoriamente un 70% de especímenes para realizar el modelo estadístico del reconocedor y un 30% para calcular la tasa de aciertos del reconocedor, cuando se intente ejecutar el programa se podrían obtener porcentajes de acierto distintos, no obstante, se ofrece una forma de poder replicar los resultados obtenidos arriba, de la siguiente forma: cuando se corra el programa, este puede recibir como argumento un número entero que represente la semilla por utilizar cuando se dividan aleatoriamente los especímenes para pruebas y para realizar el modelo estadístico, si se desean obtener los resultados anteriores basta con correr el programa de la siguiente forma: `python main.py 5`, donde `main.py` corresponde al programa que crea el modelo estadístico y determina el porcentaje de exactitud del reconocedor, y `5` corresponde a la semilla que se desea utilizar, como en el ejemplo, la semilla por utilizar será 5.

B. Otros reconocedores de caracteres y números manuscritos

1) TensorFlow con CNN

Este reconocedor de dígitos escritos a mano utiliza una red neuronal convolucional, y la base de datos MNIST, la cual tiene 60000 imágenes de dígitos escritos a mano para entrenar el modelo reconocedor y 10000 imágenes de dígitos escritos a mano para probarlo; este reconocedor de dígitos numéricos escritos a mano mostró excelentes resultados con un promedio de 92.044% de aciertos tal y como se muestra en la figura2 [1].

Entrando en detalle la librería TensorFlow implementada por Google para JavaScript, es utilizada en tareas como

Accuracy		
Class	**Accuracy**	**# Samples**
Zero	0.9672	61
One	0.9851	67
Two	0.9048	42
Three	0.9322	59
Four	0.85	40
Five	0.881	42
Six	0.88	50
Seven	0.9787	47
Eight	0.9184	49
Nine	0.907	43

Fig. 2: Aciertos TensorFlow con CNN [1]

inteligencia artificial y Machine Learning. Reúne una serie de modelos y algoritmos de Machine Learning y Deep Learning, y también permite a los desarrolladores crear gráficos de flujo de datos y estructuras que describen cómo los datos se mueven a través de un gráfico [1].

Con respecto a las redes neuronales convolucionales, estas son un poco mas complejas de entender; estas son redes neuronales que se caracterizan por aplicar una capa previamente definida la cual se conoce como kernel y a la que se le aplica una operación matemática conocida como convolución, estas se realizan de tal forma que si la capa es de 3x3, cada píxel se va a multiplicar su valor con el valor de la capa de su posición correspondiente, seguidamente se suman todas las multiplicaciones y el resultado es el valor del nuevo píxel, este procedimiento se realiza a la imagen en su totalidad. Jugando un poco con los valores de la capa o kernel se pueden obtener diferentes resultados, como por ejemplo se pueden obtener los bordes de una imagen o se le puede aplicar un blur, en la figura3 se puede apreciar un ejemplo de esto donde se utiliza un kernel 3x3 donde todos los los valores son -1 con excepción del central que es 8 [2].

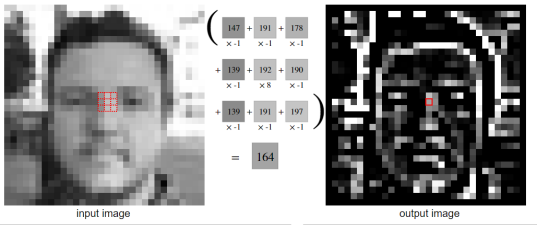

Fig. 3: Ejemplo de Convolución [2]

Lo curioso de las convoluciones es que se pueden aplicar varias veces de tal manera que la salida de una imagen sea la entrada de la siguiente convolución, esto lo que provoca es que se pierda calidad en la imagen pero la cantidad de capas de características de la misma va a aumentar, por lo que entre mas convoluciones secuenciales se apliquen mayor cantidad de características se obtendrán, es importante recalcar que en cada una de las capas se pueden aplicar diferentes kernel para obtener diferentes características; siguiendo esta teoría es como funcionan las redes neuronales convolucionales, de tal forma que lo que se entrenan son las capas convolucionales [2].

2) Keras con Perceptrón Multicapa

Esta es una biblioteca comúnmente utilizada para crear programas de reconocimiento de dígitos debido a lo fácil que es diseñar redes neuronales en ella. Para esto se emplea la red neuronal de tipo Perceptrón Multicapa.

Un perceptrón es una neurona artificial que mediante cálculos detecta características o tendencias de los datos entrantes. A estas características se le asignan valores de peso que se deben ir perfeccionando hasta que se llega a uno muy parecido al del resultado que ya se conoce. Si el peso no es ideal al final del recorrido en la red, se realiza una retro propagación que permite saber a todas las neuronas que los pesos deben seguirse ajustando.[3]

Ahora es momento de explicar como se configura la red neuronal para que empiece a aprender. Para esto primero debemos generar y formatear las muestras en las que se va a basar, para lo cual podemos usar una gran cantidad de imágenes de los distintos dígitos que se pueden obtener de la base de datos MNIST, a los cuales se les debe llevar a cabo un preprocesamiento.

Para empezar todas las imágenes deben tener las mismas dimensiones; una vez asegurado lo anterior podemos cargar los datos en varias variables, donde unas servirán de prueba y otras para entrenar a la red. Ahora se deben binarizar los colores de la imagen para que tengan valores de 0 y 1. Por otra parte, con respecto a los datos de salida que son los números del 0 al 9, se les debe realizar una matriz mediante One-Hot Encoding que nos sirva para extraer los datos de salida generados por la red.[4]

Finalmente, con los datos procesados debemos configurar la red. Lo ideal es que tenga una capa de entrada con una cantidad de neuronas igual a la de los píxeles en la imagen para analizar cada píxel por aparte. Luego se deben generar una capa oculta con la suficiente complejidad de modelo para reconocer los dígitos entrantes. Y para terminar una capa con la cantidad de salidas para generar los datos finales de cada número.[4]

Ya configurada la red solo nos queda entrenarla y con las muestras disponibles, pueden efectuarse varias iteraciones de este aprendizaje para obtener unos datos más acertados y tener una menor cantidad de fallos. Ya con la red en este estado se pueden ejecutar test para verificar su precisión. Este tipo de reconocedor tiene una tasa promedio de aciertos del 98% entrenándose con 60000 muestras y probándose con 10000 imágenes de dígitos durante 10 períodos.[4]

C. Hipótesis con base en comparación de resultados

D. Comparación con resultados de TensorFlow con CNN

- Las muestras utilizadas en este algoritmo para entrenarlo no fueron tratadas por ningún tipo de preprocesamiento debido a que la base de datos de MNIST ya viene equipada con estos dígitos listos para ser utilizados ya que previamente fueron tratados por grupos de desarrolladores profesionales, lo que implica que probablemente vienen mejor estandarizados que los que utilizamos para nuestro reconocedor.

- Las redes neuronales convolucionales son un área muy amplia y con mucha capacidad, estas redes neuronales son capaces de obtener varias características de las imágenes y posteriormente utilizar estas características para entrenar el sistema reconocedor; lo que claramente lo hace mucho mejor que el reconocedor de dígitos numéricos escritos a mano implementado para este laboratorio ya que el mismo solo obtiene una característica de las imágenes que es el histograma obtenido del conteo de los píxeles que forman parte del numero.
- La implementación de TensorFlow con CNN es mas flexible en el sentido que este algoritmo se puede ajustar a reconocer otros tipos de patrones dentro de las imágenes simplemente cambiando su kernel, lo cual es el motivo de existir de las redes neuronales convolucionales. El algoritmo utilizado para este laboratorio no es tan flexible y prácticamente solo se podría utilizar para reconocer caracteres o símbolos escritos a manos, o ir un poco mas allá y utilizarlo con imágenes que en escala de grises se puedan obtener patrones o información relevante de los píxeles con tonalidad 0 y tonalidad 255.

1) Comparación con resultados de Keras

- Nuestra cantidad de especímenes era de 6000 aproximadamente contra los 60000 utilizados por el reconocedor de Keras. Esto posiblemente ayudo mucho a que su modelo reconocedor estuviera mucho más afinado.
- Los especímenes utilizados por Keras posiblemente tenían mucha mejor calidad que los utilizados por nuestro programa. Debido a que podían darse casos en que nuestros especímenes no tuvieran la suficiente claridad porque el preprocesado no les favorecía o la calidad base ya era bastante terrible. Esto pudo darse al tener algunos especímenes tomados con una iluminación o enfoque incorrecto.
- La manera en que desarrollamos el reconocedor no tenía una manera de corregir sus modelos para obtener una mejor precisión. Los perceptrones por otra parte, si tenían la capacidad de comunicarse entre ellos para corregir los pesos que le daban a cada resultado (mediante la retro propagación) y obtener mejores resultados hasta obtener un modelo final más preciso.

II. Conclusiones

- Los histogramas en el desarrollo de modelos estadísticos reconocedores de dígitos escritos a mano son una buena herramienta que permite obtener resultados altos de alrededor de 82%, al ser un método que no es muy complejo de implementar permite crear reconocedores de forma rápida y sencilla, sin embargo, tiene inconvenientes como que la tasa de aciertos para todos los dígitos no es necesariamente uniforme, ya que mientras un dígito como el 3 se reconoce con un 67.17% de exactitud, el dígito 5 se reconoce con una gran tasa de acierto de 91.92%, por ende se algunas recomendaciones que podrían hacer el reconocedor más efectivo son, generar el modelo estadístico con más especímenes ya que actualmente se utilizan 465 imágenes de cada dígito para crearlo, la segunda recomendación es analizar la obtención/extracción de mayor información, es decir, ampliar el vector de características para que contenga más información aparte de la que se obtiene del histograma.
- Una parte del programa a la que quizás no se le da la suficiente importancia es la de obtener especímenes de calidad para que el reconocedor pueda tener buenas bases en las que generar su modelo. Es cierto que la cantidad es importante, pero el resultado obtenido va a ser malo si las pruebas no tienen un estándar mínimo de calidad, que las haga lo suficientemente complejas como para obtener los datos, sus datos específicos que las diferencien y se pueda lograr una separación clara de los distintos casos de dígitos.
- Tomando en cuenta que el histograma de los dígitos pueden ser muy similares a pesar de que a simple vista estos no se parezcan en nada, y que este reconocedor solamente puede diferenciar 10 caracteres numéricos escritos a mano; podemos decir que este algoritmo es útil para este caso en específico, sin embargo, debido a que los histogramas en algunos casos pueden llegar a ser muy similares y si aumenta el número de posibles respuestas del reconocedor, este algoritmo no sería tan útil, por lo que sería mejor optar por otro tipo de algoritmo en caso de que el número de posibles respuestas sea mayor y se tengan que comparar los histogramas de muchos caracteres no solo numéricos.

References

[1] "TensorFlow.js: Reconocimiento de dígitos escritos a mano con CNN," S.F. [Online]. Available: https://codelabs.developers.google.com/codelabs/tfjs-training-classfication?hl=es-4190

[2] "CS231n Convolutional Neural Networks for Visual Recognition," s.f. [Online]. Available: https://cs231n.github.io/convolutional-networks/

[3] "Perceptrón: ¿qué es y para qué sirve?" s.f. [Online]. Available: https://datascientest.com/es/perceptron-que-es-y-para-que-sirve

[4] "RECONOCIMIENTO DE DÍGITOS (NÚMEROS) MANUSCRITOS," s.f. [Online]. Available: https://unipython.com/reconocimiento-de-digitos-numeros-manuscritos/