Identificador de números en base a imágenes

Neil Otniel Moreno Rivera
Universidad de Guanajuato, no.morenorivera@ugto.mx

***Resumen*— El problema desarrollar un programa, que identifique mediante la base de datos MNIST, una serie de números en fotografía mediante análisis de pixeles en escala de grises se desarrolló mediante una red neuronal.**

***Abstract*-- The problem is to develop a program that identifies, through the MNIST database, a series of numbers in photography through grayscale pixel analysis, was developed using a neural network.**

I. Introducción

Primero debemos saber de donde sacaremos los datos, para el entrenamiento y su posterior prueba. Esta es una enorme base de datos de dígitos escritos manualmente que se utiliza normalmente para preparar diferentes sistemas de manejo de imágenes, la base de datos del MNIST contiene 60.000 imágenes de preparación y 10.000 imágenes de prueba. Estos datos se entregan de manera de un arreglo, haciendo arreglos de ventiocho por ventiocho, estos representando las imagenes, como un mapa de grises, donde se hace notar el numero en blanco (Un gris más claro en la figura 1) esto además de un identificador que sirve vaya la redondancia para identificar y saber el número que se encuentra plasmado en la imagen.

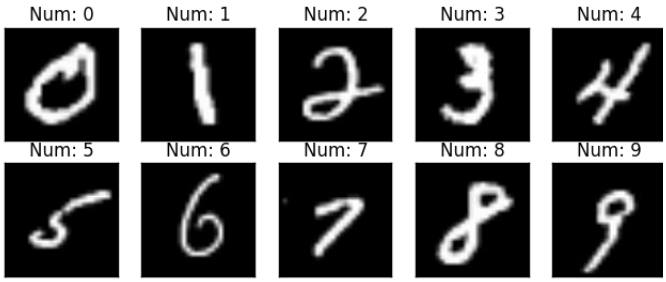

Fig. 1. Imagen de ejemplo, de los números proporcionados por MNIST.

II. Teoria

Para la resolución de este problema, se encontró una forma muy sencilla para cargar los datos, puesto que el lenguaje de programación conocido como Python, ya nos permite conseguir todos los datos de la base de datos, solamente importándolos, por lo que es un problema menos, ahora solo debemos desarrollar el sistema que interprete esos datos, de estos datos tomaremos un 80% para el entrenamiento y un 20% para probarlo y conocer una aproximación correcta.

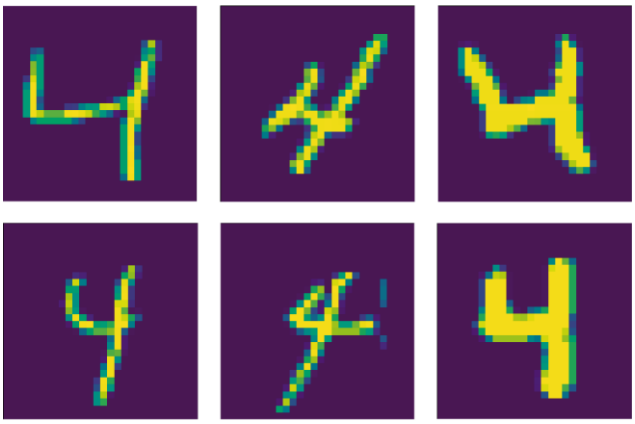

Fig 2. Imagen de ejemplo de los números proporcionados por MNIST.

Ahora con estos datos, lo que se realizo fue realizar un aplanado para que estos por arreglo dieran como resultado, un vector de 1 x 784, con este vector ya nos era posible introducirlo en una red neuronal, pero para que pudiera identificar los resultados correctamente, teníamos que tomar en cuenta sus etiquetas, por lo que fueron cargadas en una variable para después ser usadas como categorías, que fueron asignadas a las salidas de la red neuronal, en este caso 10, por lo que cada una coincidían con los números desde el 0 hasta el 9, por lo que la salida estaba clara.
Para el modelo se utilizaron 20 épocas (épocas, refiriéndose al número de iteraciones que fueron tomadas) con un tamaño de 128, también se normalizaron los datos, dividiendo por 255, para que solo los dígitos importantes tuvieran un valor y no hubiera una separación tan grande en los datos porque los valores iban desde 0 hasta 255, esto haciendo que la precisión llegue a bajar considerablemente (Viéndolo como el más oscuro como ejemplo de la figura 3).

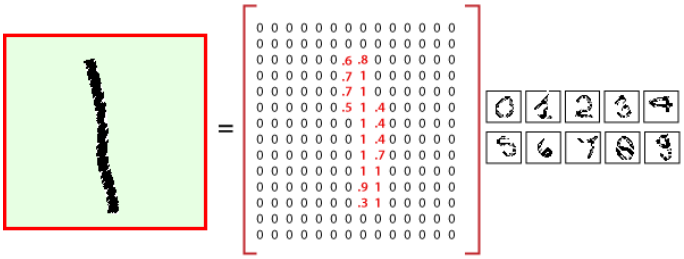

Fig 3. Imagen con los valores significativos de una imagen con un uno contenido.

Para la red neuronal se siguieron los parámetros donde se buscaba un mejor funcionamiento y ahorro de recursos, esto

* Neil Otniel Moreno Rivera.

porque al principio se decidió tener dos capas ocultas con una cantidad absurda de neuronas en cada una, para asegurar un resultado acertado, aproximadamente 500 y 200, pero para economizar recursos, se probó con únicamente 50 y 20, en las capas ocultas, aparte de una capa de salida con 10 neuronas para representar las categorías dichas anteriormente.

III. Resultados

Los resultados obtenidos, fueron claros, aunque un tanto extraños, puesto que teníamos en consideración que por la cantidad de neuronas o el tipo de desarrollo tan básico que se iba a realizar, se podía obtener una precisión máxima del 70%, sorprendiéndonos con una precisión del 98.62%, esto siendo comprobado después por una matriz de confusión, siendo utilizadas las mismas 10 categorías, con pérdidas bastante pequeñas comparadas con los datos de prueba introducidos (20%), para que la matriz de confusión este correcta esta debe tener mayor cantidad de datos en la diagonal principal, de otra forma se considerarían errores.

Como comprobación del resultado podemos usar la siguiente matriz de confusión:

```
Confusion matrix:
 [[1368    0    2    1    4    1    6    0    3    2]
 [   0 1558    5    1    2    0    3    4    5    2]
 [   4    5 1399    4    8    3    7    5    7    1]
 [   2    1   25 1364    0   22    0    6   10    5]
 [   0    1    4    0 1312    2    6    6    2   17]
 [   3    2    7   19    2 1168   16    0    7    7]
 [   5    1    4    1    6    6 1362    1    1    0]
 [   2    8   24    8    8    1    1 1388    2   16]
 [   1   13    7    9    5    5    4    1 1310   13]
 [   4    0    1   14   27    3    1   10   14 1287]]
```

Fig 4. Imagen que muestra la matriz de confusión.

La figura 4, muestra la matriz de confusión que obtenemos al evaluar los datos, esta matriz se ve bastante buena, dando más del 95% de precisión esto se sabe porque al probar los datos con los datos guardados, obtenemos una precisión del 96.54% por lo que el modelo funciona bastante buen o al menos en bastantes situaciones podría reconocer los números.

IV. Concluciones

Para concluir con esto, podemos decir que el modelo desarrollado funciona de una manera más que aceptable, para identificar números escritos a mano, esto tomando en cuenta la base de datos proporcionada de MNIST, y gracias también a su preprocesamiento o normalización de los datos, para su mejor aprovechamiento.

```
Test loss: 0.11022443324327469
Test accuracy: 0.9654285907745361
```

Fig 5. Imagen que muestra un aproximado de los datos de prueba perdidos.

La figura 5, representa los datos perdidos y los datos correctos, esto podríamos interpretarlo de una manera favorable, puesto que en la mayoría de los casos funciona el modelo desarrollado. Estos datos se encuentran también representados en la matriz de confusión de la figura 4.

Bibliografia

[1] Sankar Pal, (2000) Soft Computing for image processing, Springer.