Redes neuronales multicapa y redes neuronales convolucionales aplicadas al reconocimiento de dígitos manuscritos

1er Yamil Dylan Aucca Quispe
Departamento de Ciencias de la Computación, Universidad Nacional de Ingeniería, Lima, Perú
yamil.aucca.q@uni.pe

***Resumen*—Este trabajo presenta la implementación de una red neuronal multicapa (MLP) y una red neuronal convolucional (CNN) para el reconocimiento de dígitos escritos en una página web donde se pueden dibujar dígitos en un recuadro para luego presentar la predicción presentada por los dos tipos de modelos aplicados. Se utilizó el conjunto de datos MNIST para el entrenamiento de ambas arquitecturas de inteligencia artificial (aprendizaje supervisado). Además, se utilizaron las funciones de activación ReLU y Softmax, así como también la función de costo categorical cross-entropy y el algoritmo de optimización Adam para el descenso de gradiente. Se espera determinar cuál de las dos arquitecturas artificiales es la más eficiente para la tarea de clasificar imágenes de dígitos escritos. Las redes neuronales se realizaron en Python con el apoyo de librerías como TensorFlow, Numpy y Matplotlib. Para la página web se utilizó HTML, CSS y JavaScript.**

***Index Terms*—aprendizaje profundo, redes neuronales multicapa, redes neuronales convolucionales, MNIST**

I. Introducción

Las redes neuronales multicapa (MLP) comenzaron a desarrollarse en el siglo XX con el objetivo de abordar problemas que los perceptrones no podían resolver. A lo largo del tiempo, estas redes han evolucionado significativamente, dando lugar a diversos tipos de modelos de inteligencia artificial, como las redes neuronales convolucionales (CNN). [1]

Ambas arquitecturas de aprendizaje profundo han sido ampliamente utilizadas en diversas tareas dentro del campo de la visión por computadora, gracias a su eficiencia y a su superioridad en precisión en comparación con otras soluciones. [2] Una tarea destacada en la que se emplean estas redes neuronales es el reconocimiento de dígitos manuscritos.

Este documento presenta la implementación de dos tipos de redes neuronales (multicapa y convolucional) las cuales serán de clasificación pues al utilizar el conjunto de datos MNIST para el reconocimiento de dígitos manuscritos se buscará dar como resultado la predicción sobre cuál es el dígito escrito. Además, como MNIST contiene datos etiquetados entonces el aprendizaje será de tipo supervisado. Finalmente, se busca obtenerla precisión alcanzada por cada una de las arquitecturas artificiales implementadas para con ello determinar cuál de los dos tipos de redes neuronales usadas es más adecuada para el reconocimiento de dígitos manuscritos.

II. Fundamentos teóricos

II-A. Redes neuronales multicapa (MLP)

Las redes neuronales multicapa (MLP) se componen de tres tipos de capas: una que recibe los datos de entrada (input layer), una o más capas que realizan diversos cálculos sobre los datos (hidden layers), y una capa que devuelve los resultados de las operaciones (output layer). Cada una de estas capas está formada por neuronas (elementos que almacenan números) que están conectadas entre sí y tienen asignados pesos (weights) que determinan la importancia de las conexiones. Además, cada neurona poseen un sesgo (bias) el cual se adiciona a la suma ponderada de los pesos y los valores de la capa anterior, finalmente, se aplica al resultado una función de activación. [3]

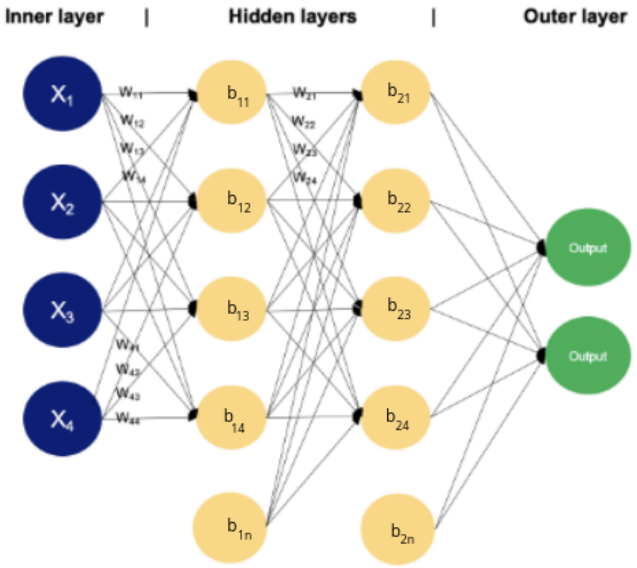

Figura 1. Arquitectura de una red neuronal multicapa con 2 hidden layers.

$$z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j \tag{1}$$

Con respecto a la capa j, donde z_j sería el valor ponderado (o logit) de la neurona j en la capa l. x_i es la entrada de la neurona i de la capa anterior $l-1$. w_{ij} viene a ser el peso que conecta la neurona i de la capa $l-1$ con la neurona j de la

capa l. b_j es el sesgo de la neurona j en la capa l. Finalmente, n sería el número de neuronas en la capa anterior $l-1$.

II-B. Descenso de gradiente

El descenso de gradiente (gradient descent) es un algoritmo de optimización utilizado para entrenar modelos de inteligencia artificial, como las redes neuronales, con el objetivo de minimizar los errores de la función de costo (cost or loss function). Este proceso implica ajustar los pesos del modelo en la dirección opuesta al gradiente de la función de costo, permitiendo que el modelo mejore su precisión a medida que se entrena [8]. El tamaño de cada paso (ajuste de los pesos y sesgos) se ve determinado por la taza de aprendizaje o el learning rate (η), que controla la velocidad de aprendizaje.
El gradiente de una función de costo J con respecto a los parámetros (w_{ij} y b_j) se define como:

$$\nabla J = \left(\frac{\partial J}{\partial w_{ij}}, \frac{\partial J}{\partial b_j} \right) \tag{2}$$

Donde $\frac{\partial J}{\partial w_{ij}}$ y $\frac{\partial J}{\partial b_j}$ representan las derivadas parciales de J con respecto a cada peso y sesgo, indicando la dirección de mayor incremento de la función de costo.

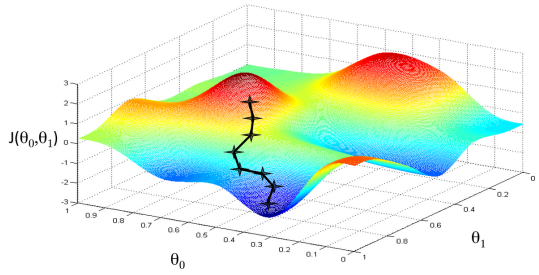

Figura 2. Ejemplo de un descenso de gradiente aplicado a la superficie generada por una función de costo.

Uno de los optimizadores más utilizados es Adam (Adaptive Moment Estimation), que combina las ventajas de los algoritmos AdaGrad y RMSProp. [9]

II-C. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales (CNN) son un tipo de modelo de inteligencia artificial que se basa en las redes neuronales multicapa. Al igual que las MLP, las CNN tienen una capa de entrada (input layer) y una capa de salida (output layer). Sin embargo, sus capas ocultas (hidden layers) se clasifican generalmente en tres tipos: Capa de convolución, capa de pooling y capa densa (fully connected layer). [4] Estas capas se encuentran en la siguiente sucesión generalmente: Convolución → Pooling → Convolución → Pooling, y así sucesivamente.

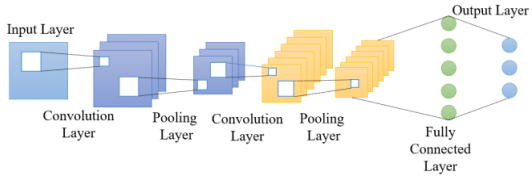

Figura 3. Arquitectura general de una red neuronal convolucional.

II-C1. Capa de convolución: En la capa de convolución se realiza la operación de convolución, la cual consiste en aplicar diferentes filtros (kernels) para cada uno de los 3 canales de una imagen (RGB, los cuales corresponden a los colores de cada pixel) con el fin de obtener una nueva imagen. Este proceso permite extraer un conjunto de características que ayudan a identificar patrones (bordes o texturas) en las imagenes, generando lo que se conoce como un mapa de características (feature map). Matemáticamente, mediante los kernels y cada canal de una imagen (ambos como matrices) se realiza el producto punto entre ambas por cada [5]

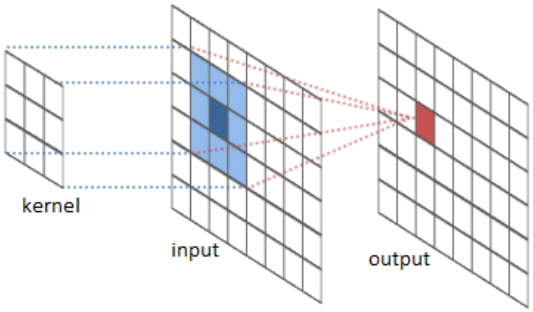

Figura 4. Operación de convolución.

II-C2. Capa de pooling: En la capa de pooling se lleva a cabo la operación de pooling, que se realiza después de la convolución. El objetivo de esta operación es reducir las dimensiones del mapa de características generado, lo que a su vez disminuye la carga computacional y ayuda a prevenir el sobreajuste. [6] Hay dos tipos de pooling muy utilizados, el primero es el max pooling donde se busca el mayor valor en una región (conocida como ventana)

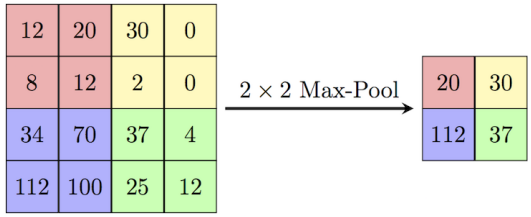

Figura 5. Operación de max pooling en una ventana de 2x2.

Mientras que en el average pooling se busca el promedio de los valores contenidos en una ventana.

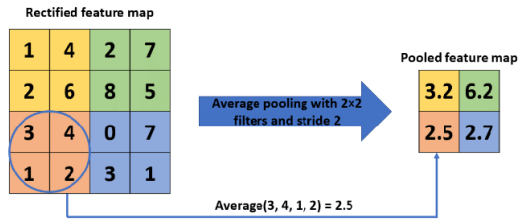

Figura 6. Operación de average pooling en una ventana de 2x2.

II-C3. Capa densa (fully connected layer): Son capas que se utilizan tanto para las redes neuronales multicapa (como sus hidden layers, principalmente) así también como para las convolucionales (luego de realizar los procesos de convolución y pooling, para manejar los valores numéricos obtenidos).

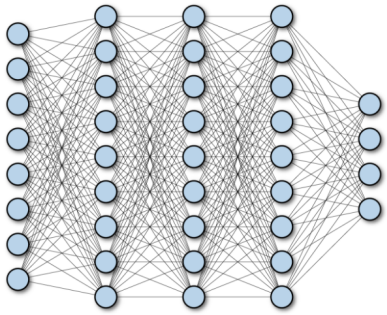

Figura 7. Modelo de 5 capas densas (fully connected layers).

II-D. Función de activación

Las funciones de activación son fundamentales para introducir no linealidades en las redes neuronales, lo que permite encontrar relaciones más complejas entre los datos. Sin estas funciones, una red neuronal, sin importar cuántas capas tenga, se comportaría como una simple combinación lineal de las entradas. Algunas de estas funciones son: ReLU y Softmax. [7]

$$f(x) = \text{máx}(0, x) \quad \text{(ReLU)} \tag{3}$$

Donde x viene a ser el valor calculado en una neurona antes de pasar por la función de activación y $f(x) \in [0, \infty)$

$$g(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad \text{(Softmax)} \tag{4}$$

Donde z_i sería el logit para la clase i (en el caso de MNIST, sería para una clase específica correspondiente a un dígito). Mientras que z_j sería el logit para la clase j (en el caso de MNIST y al encontrarse el logit dentro de una sumatoria, se recorre cada clase correspondiente a un dígito). Finalmente, $g(z_i) \in [0, 1]$.
Cabe mencionar que ReLU es utilizada en las hidden layers debido a que los valores que retorna poseen no linealidad apoyando a un aprendizaje más complejo de la red neuronal, mientras que Softmax solo es utilizada para la output layer porque los valores que retorna son de carácter probabilístico, ya que al estar entre 0 y 1 pueden interpretarse como la probabilidad de que una entrada pertenezca a una clase o no.

II-E. Función de costo (cost or loss function)

La función de costo nos permite evaluar cuán eficiente es nuestra red neuronal al comparar las predicciones realizadas con los resultados reales. Esta función es crucial para el entrenamiento del modelo, ya que guía el proceso de optimización.

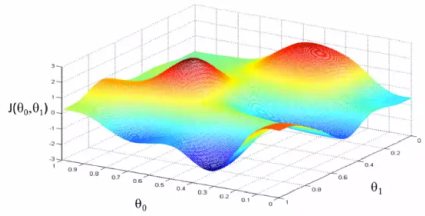

Figura 8. Gráfica 3D de una función de costo.

Entre las que son utilizadas tenemos a la función categorical cross-entropy (existen otras más como la función mean squared error, hinge loss, etc. pero la mostrada resulta especial para problemas de clasificación):

$$H(p, q) = -\sum_{i=1}^{C} p_i \cdot \log(q_i) \tag{5}$$

Donde p_i es el valor de la etiqueta real en la posición i (para la clase correcta será 1 y 0 para las demás). Mientras que q_i es la probabilidad predicha para la clase i por el modelo. Finalmente, C viene a ser el número de clases posibles (en el caso de MNIST, serán 10 clases por haber 10 dígitos).
Es importante mencionar que dependiendo de nuestra elección entre utilizar todo el conjunto de datos o solo un subconjunto (batch o minibatch) nuestro proceso de entrenamiento puede variar según el enfoque mediante los parámetros que nosotros querramos que posea la solución a nuestro problema (precisión deseada, velocidad de entrenamiento, memoria a utilizar, etc.)

II-F. Retropropagación

Es un algoritmo fundamental que permite entrenar redes neuronales, se aplica después de utilizar la función de costo porque permite actualizar eficientemente los pesos de la red mediante el cálculo de gradientes haciendo luego uso del descenso de gradiente. En la retropropagación (backpropagation) se utiliza principalmente la regla de la cadena:

$$\frac{\partial \mathcal{L}}{\partial z^{(l)}} = \frac{\partial \mathcal{L}}{\partial z^{(l+1)}} \cdot \sigma'(z^{(l+1)}) \cdot w^{(l+1)} \tag{6}$$

Esta ecuación representa la retropropagación para capas anteriores l, donde $\mathcal{L}$ representa a la función de costo, $w^{(l+1)}$ es el vector que contiene a los pesos de la capa $l+1$, σ' viene a ser la derivada de la función de activación y $z^{(l+1)}$ es la combinación lineal o suma ponderada de la capa $l+1$.

$$\frac{\partial \mathcal{L}}{\partial w^{(l)}} = \frac{\partial \mathcal{L}}{\partial z^{(l)}} \cdot a^{(l-1)T} \tag{7}$$

Esta fórmula es esencial para calcular el gradiente de los pesos en cada capa. Donde $\frac{\partial \mathcal{L}}{\partial z^{(l)}}$ es el gradiente de la capa l, $a^{(l-1)T}$ es la transpuesta de la activación de la capa anterior $l-1$.

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \frac{\partial \mathcal{L}}{\partial z^{(l)}} \tag{8}$$

Por último, esta fórmula es esencial para calcular el gradiente de los sesgos en cada capa. Donde $\frac{\partial \mathcal{L}}{\partial z^{(l)}}$ es el gradiente de la capa l, $b^{(l)}$ son los sesgos de la capa l.

III. Aplicaciones de un modelo de inteligencia artificial capaz de reconocer dígitos manuscritos

Una red neuronal que sea capaz de reconocer dígitos manuscritos puede ser útil para la creación de diferentes tipos de programas como lo podría ser una aplicación móvil orientada a la educación en niños sobre la escritura de números.

Figura 9. Niño escribiendo números en una pizarra.

Así como también puede ser utilizada en el sector empresarial buscando acelerar el proceso de diversos valores numéricos que se encuentran en todo tipo de documentos y que necesiten ser leídos automáticamente.

Figura 10. Documento que posee datos financieros.

Además, en el caso de las encuestas o formularios que se realizan a mano y puedan contener algún número en el apartado de respuestas, para leer y procesar tales valores, la inteligencia artificial puede ser de gran ayuda.

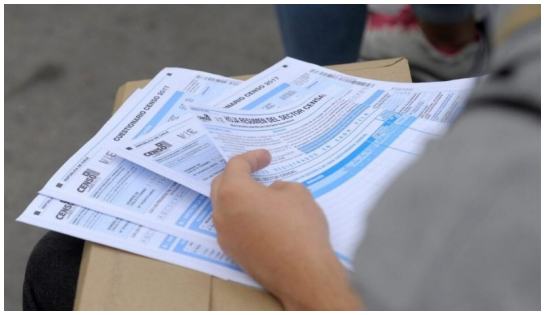

Figura 11. Ejemplo de una encuesta realizada a mano.

IV. Metodología

Sobre las redes neuronales, estas se realizarán con el lenguaje de programación Python con el apoyo de módulos como TensorFlow para la implementación de las redes neuronales, Numpy para el manejo de estructuras matemáticas y Matplotlib para la elaboración de gráficas.

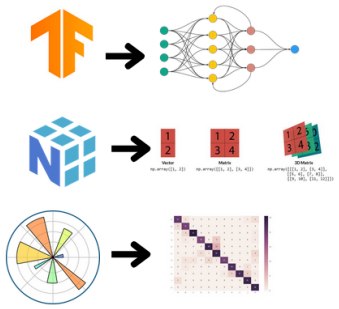

Figura 12. TensorFlow, Numpy y Matplotlib (de arriba a abajo) y los usos que se les da.

El dataset a utilizar será MNIST, la cual es una base de datos que contiene unas 60000 y 10000 imagenes etiquetadas de digitos manuscritos para el entrenamiento y testeo de una inteligencia artificial que pueda identificarlos, respectivamente. [10] En este proyecto, será utilizado tanto para las redes neuronales multicapa como para las convolucionales.

Figura 13. Imagenes contenidas en el conjunto de datos MNIST

Para la página web, se espera realizar una página con HTML (para la estructura), CSS (para el diseño) y JavaScript (para la implementación de los modelos) que se ejecute en el local teniendo la siguiente estructura:

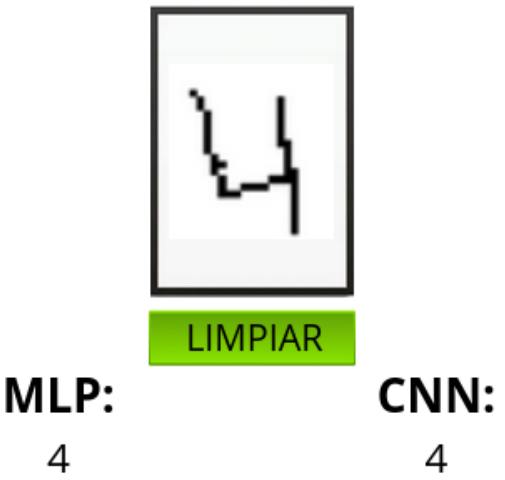

Figura 14. El centro de la página un recuadro donde se puede dibujar dígitos, un botón debajo del cuadro que permite limpiar lo dibujado y abajo se encuentra la predicción que arroja cada uno de los modelos artificiales entrenados según lo dibujado.

V. Implementación y resultados

Se realizó la implementación, mediante Python, sobre todo lo relacionado al manejo del dataset MNIST (descarga, preprocesamiento y guardado), la creación y el entrenamiento de las redes neuronales (MLP y CNN) las cuales poseen las siguientes características:

Red neuronal multicapa (MLP)

- Input layer: Capa de 784 unidades (para almacenar el valor de cada pixel de las imagenes de 28x28 de MNIST)
- Hidden layers: 3 capas densas de 256, 128 y 64 neuronas (en ese orden) y cada una antes precedida por una función de activación ReLU.
- Output layer: 1 capa densa de 10 neuronas (para recibir las probabilidades sobre a que clase o dígito corresponde la imagen) precidda por una función de activación Softmax.

Red neuronal convolucional (CNN)

- Input layer: Capa que recibe los valores de 1 solo canal de una imagen de 28x28 pixeles (la imagen se encuentra en escala de grises)
- Hidden layers: 2 capas de convolución de 32 y 64 kernels de 3x3 en ese orden sucedidas por la función de activación ReLU; ambas capas seguidas (cada una) por una capa de max pooling de kernel 2x2. Luego, una capa que sirve como aplanado (flatten) de los valores de la imagen procesada y una última capa densa de 128 neuronas precedida por la función de activación ReLU.
- Output layer: 1 capa densa de 10 neuronas (para recibir las probabilidades sobre a que clase o dígito corresponde la imagen) precidda por una función de activación Softmax.

La arquitectura de cada una de las redes neuronales a crear y entrenar corresponde a la necesidad de buscar que cada una pueda ser entrenada eficientemente. Para la MLP, la reducción gradual de la cantidad de neuronas en las hidden layers permite que el modelo vaya aprendiendo relaciones cada vez más abstractas y simplificadas de los datos de entrada. Mientras que para la CNN, el uso del max pooling corresponde a la búsqueda de encontrar patrones representativos o resaltantes en la entrada (bordes, esquinas, etc.) y el aumento gradual de los kernels en las capas de convolución corresponde a la necesidad de aprender relaciones más abtractas y complejas de las imagenes a medida que va pasando por la red neuronal. Es importante mencionar que cada imagen ha sido procesada de tal forma que cada valor correspondiente a un pixel depende del color siendo que 0 corresponde al negro y 1, al negro.

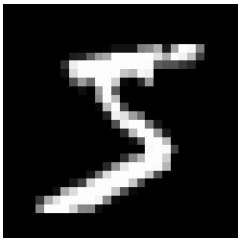

Figura 15. Imagen perteneciente al dataset MNIST (dígito 5)

Luego, todas las imagenes preprocesadas fueron utilizadas para el entrenamiento de las redes neuronales utilizando como función de costo a categorical crossentropy, como optimizador a Adam y como parámetros los siguientes: Tamaño del batch: 32; épocas de entrenamiento: 10 y como tasa de aprendizaje (learning rate): 0.001.

Finalmente, luego del entrenamiento realizado a los dos modelos artificiales, se consigue la siguiente métrica de precisión: **99.21 % para la MLP y 99.85 % para la CNN**. Esto también puede apreciarse en las matrices de confusión de cada red neuronal.

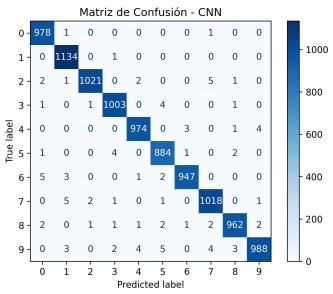

Figura 16. Matriz de confusión de la red neuronal convolucional entrenada

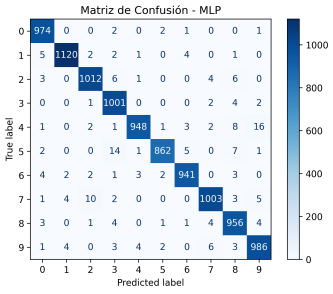

Figura 17. Matriz de confusión de la red neuronal multicapa entrenada

En estas gráficas resulta evidente como la red neuronal convolucional predice con mayor precisión todos los dígitos (del 0 al 9) en comparación a la red neuronal multicapa. Pues esta última, si bien es menos precisa en la predicción de todos los dígitos que la CNN, se puede percibir como sufre a la hora de predecir dígitos como el 4, 5 y 7. Posiblemente, esto debido a las características de estos números que pueden ser similares a otros, de ahí surge la confusión y, por lo tanto, el error en la predicción.

Por último, con los modelos ya entrenados, se busca su implementación en una página web la cual es creada a través de HTML, CSS y JavaScript (para el frontend) donde HTML sirve para la estructura, CSS, el diseño y JavaScript, la lógica interactiva en la página. Todo ello se encuentra trabajando en una pequeña aplicación web realizada con Flask, la cual permite el funcionamiento de las predicciones de los modelos entrenados.

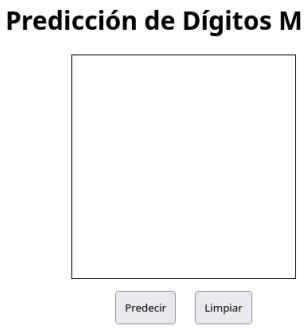

Figura 18. Visualización de la página web creada para la implementación de las redes neuronales (MLP y CNN)

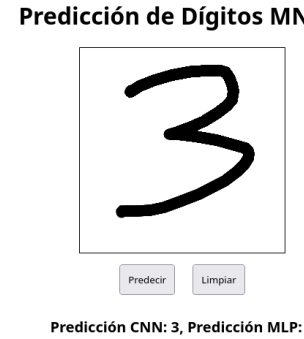

Figura 19. Visualización de la página web en funcionamiento, luego de presionar el botón ”Predecir”, el dígito dibujado pasa como entrada a los modelos entrenados (después de un preprocesamiento donde se ajusta el tamaño y se invierten los colores) y, finalmente, las predicciones son realizadas

VI. Conclusiones y desarrollo posterior

- Se determina que la red neuronal convolucional resulta más eficiente para la tarea de reconocer dígitos manuscritos, que una red neuronal multicapa (99.21 % para la MLP y 99.85 % para la CNN, de precisión). Esto debido a que la entrada que reciben son cada canal de una imagen (en este caso solo un canal pues tratamos imagenes en escala de grises) y se trata directamente con los datos en su forma de matriz, permitiendo esto el poder resolver de manera mucho más compleja y precisa un problema que abarca el uso de imágenes, las cuales están representadas como matrices, para una inteligencia artificial.
- Se crearon y entrenaron las redes neuronales (MLP y CNN) con el dataset MNIST, para luego posteriormente implementarse los modelos ya entrenados en una página web. Sin embargo, queda pendiente el seguir optimizando el proceso de entrenamiento de las redes neuronales con el fin de obtener mejores resultados en las predicciones (mayor precisión), así como mejorar la velocidad del entrenamiento.
- Se espera mejorar a futuro el funcionamiento de la aplicación web con el fin de que pueda mandar, de manera adecuada, la imagen del dígito dibujado (sin importar la posición donde se coloque) con el fin de que las predicciones puedan ser realizadas por los modelos artificiales entrenados de manera correcta.

Referencias

[1] I. Goodfellow, Y. Bengio, y A. Courville, *Deep Learning*, Cambridge, MA, USA: MIT Press, pp. 220-223, 2016.
[2] IBM, ”What is deep learning?”, [En línea]. Disponible: https://www.ibm.com/es-es/topics/deep-learning. [Accedido: 10 octubre 2024].
[3] K. Y. Chan, B. Abu-Salih, R. Qaddoura, A. M. Al-Zoubi, V. Palade, D.-S. Pham, y J. Del Ser, “Deep neural networks in the cloud: Review, applications, challenges and research directions,”*Neurocomputing*, vol. 545, pp. 3-4, 2023.
[4] L. Alzubaidi, J. Zhang, A. J. Humaidi, y otros, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,”*Journal of Big Data*, vol. 8, p. 53, pp. 14-19, 2021.
[5] R. Yamashita, M. Nishio, R. K. G. Do, y otros, “Convolutional neural networks: an overview and application in radiology,”*Insights Imaging*, vol. 9, pp. 613-614, 2018.
[6] F. Bieder, R. Sandkühler, y P. C. Cattin, “Comparison of Methods Generalizing Max- and Average-Pooling,.ªrXiv preprint arXiv:2103.01746, pp. 3-4, 2021.
[7] S. R. Dubey, S. K. Singh, y B. B. Chaudhuri, “Activation functions in deep learning: A comprehensive survey and benchmark,”*Neurocomputing*, vol. 503, pp. 92-99, 2022.
[8] IBM, “What is gradient descent?”, [En línea]. Disponible: https://www.ibm.com/topics/gradient-descent. [Accedido: 18 octubre 2024].
[9] D. P. Kingma y J. Ba, “Adam: A Method for Stochastic Optimization,”*CoRR*, vol. abs/1412.6980, 2014.
[10] Y. LeCun, C. Cortes, y C. J. C. Burges, “The MNIST database of handwritten digits”, [En línea]. Disponible: https://yann.lecun.com/exdb/mnist/. [Accedido: 10 octubre 2024].