PROYECTO FINAL:
RECONOCIMIENTO DE DÍGITOS MEDIANTE REDES NEURONALES CONVOLUCIONALES.

Santiago Ortiz Suarez, Johan Estiven Robayo Linares, Sharon Daniela Robayo Niño
Cod: 160004034, 160004040, 160004041.
Programa: *Ingeniería de sistemas.*
Facultad: *Ciencias Básicas e Ingeniería*
Email:
santiago.ortiz.suarez@unillanos.edu.co
johan.robayo@unillanos.edu.co
sharon.robayo@unillanos.edu.co

***Resumen* —** En el presente informe se describe las definiciones matemáticas y el funcionamiento en código del proyecto final del curso de optimización dictado por el ingeniero electrónico Héctor Miguel Vargas García, el cual consistió en implementar una red neuronal convolucional en una aplicación de escritorio desarrollada en el lenguaje Python, para el reconocimiento de dígitos escritos a mano con el uso del mouse a través de una interfaz gráfica.

Palabras claves— Red neuronal convolucional, keras, reconocimiento de imágenes, función de activación, optimizador, capa oculta, capa densa.

I. INTRODUCCIÓN

Las redes neuronales convolucionales. Éstas, son un tipo de red neuronal artificial donde las neuronas se corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico. Por lo que son muy efectivas en el campo de la visión artificial, cuya aplicación más destacada es el reconocimiento y clasificación de imágenes o de patrones en la imagen [1].

Para entender las redes neuronales convolucionales es necesario comprender que la convolución es una operación matemática, en la que una función se "aplica" a otra función, el resultado se puede entender como una "mezcla" de las dos funciones, las convoluciones son realmente buenas para detectar estructuras sencillas de una imagen, y esas sencillas funciones juntas, permiten construir funciones aún más complejas. En una red convolucional, este proceso ocurre sobre una serie de muchas capas, en la que cada una de ellas realiza una convolución sobre el resultado de la capa anterior [2].

La estructura de una red neuronal convolucional se puede apreciar en el ejemplo indicado en Fig. 1.

II. EXPERIMENTOS Y RESULTADOS

A. Funcionamiento interno de las librerías implementadas.

Inicialmente se toma como referencia la capa Conv Layer de la estructura de una red neuronal convolucional (ver Fig 2). En esta capa se usa la operación matemática convolución, la cual, se representa como el producto entre una imagen de entrada representada como X y un filtro representado con f , entonces la expresión estará dada por eq(1).

La dimensión de entrada de la imagen X es nxn, para la representación del filtro f, la dimensión debe ser cuadrada y menor a las dimensiones de X, y sus valores son binarios y aleatorios. Además, la dimensión de la convolución Z es definida por eq(2) como se muestra en Fig 3.

Posteriormente se pasa a la capa completamente conectada (Fully Connected Layer), en esta capa se necesita el vector X en una dimensión debido a que posteriormente se realiza una transformación lineal, para esto se transpone cada fila de manera ascendente, donde estas se apilan de manera descendente para crear un vector columna como se puede apreciar en Fig 4.

Seguidamente, se realiza una transformación lineal teniendo en cuenta la estructura de la red neuronal (ver Fig 5), definida por eq(3), donde X es la imagen de entrada, la cual fue redimensionada anteriormente, W es una matriz que define el peso asociado a cada neurona (n), donde sus columnas están definidas por n y el número de filas por 2n, b es un vector cuyas dimensiones dependen de la cantidad de neuronas. Los valores de W y b se definen de manera aleatoria como se definen en Fig 6.

Después, se elige una determinada función de activación, la cual será utilizada para aplicar una transformación no lineal a los datos de salida de la capa inmediatamente anterior. Esta función de activación se agrega en cada capa de la red neuronal, varía según el problema a resolver, por ejemplo: Para problemas de clasificación binaria suele utilizarse la función de activación sigmoide, la cual está definida por eq(4). Para el caso del proyecto, al tratarse de reconocimiento de imágenes, en las

tres primeras capas se utilizó la función de activación ReLU, debido a que tiene un buen desempeño en redes convolucionales, está función no está acotada, como consecuencia pueden morir demasiadas neuronas y su activación es de tipo sparse (solo se activa si los valores de entrada son positivos), su ecuación está dada por eq(5). En la última capa del modelo neuronal del proyecto se utilizó la función de activación softmax, ya que esta se utiliza para normalizar los datos de entrada, está acotada entre 0 y 1, y tiene un buen rendimiento en las últimas capas, su fórmula está dada por eq(6).

Las capas encargadas de calcular la convolución, la función de activación sigmoide y la transformación lineal, son denominadas capas ocultas, como se muestra en (Fig 7), estas a su vez están inmersas en capas superiores llamadas capas densas. Las capas densas se encargan de simplificar el modelo para obtener una estructura más entendible, mediante la especificación de la función de activación y el número de neuronas.

A continuación se procede a calcular la regresión no lineal a partir del resultado obtenido en eq(3), donde estos datos conforman la entrada a la capa donde se aplica la función de activación (ver eq(7)). Se procede a aplicar la transformación lineal descrita por eq(8), donde A es el resultado de la aplicación de la función de activación en la capa oculta inmediatamente anterior. Finalmente, la salida de las capas densas es la aplicación de la función de activación a los datos de entrada de la penúltima capa oculta, como se muestra en eq(9) [4].

Por otro lado, en las redes neuronales convolucionales se utiliza una función denominada *optimizer*, la cual es un algoritmo que ayuda a minimizar o maximizar la función objetivo dependiendo del caso, a partir del conjunto de datos de entrada X. Esta función ayuda al proceso de aprendizaje mediante la actualización de la dirección de la solución óptima, es decir, minimizando la pérdida en el proceso de formación de la red, lo cual hace que esta función tenga un peso importante dentro de la estructura de la red neuronal.

Hay muchos algoritmos que se encargan de optimizar el proceso de aprendizaje, para este caso en especial, se usó la librería keras, la cual tiene los siguientes algoritmos: Gradiente descendiente estocástico (SGD), Adam, Adadelta, Adamax, Adagrad, Nadam, entre otros. Para el proyecto se seleccionó Adam, cuya ecuación está dada por eq(10).

Donde α indica el tamaño de paso cuyo propósito es actualizar el avance que tiene el algoritmo para evitar cambios bruscos, $\hat{v}_t$ representa el factor adaptativo del descenso, $\hat{p}_t$ es la dirección de descenso, ε es una constante que tiende a cero. X_t es el punto actual X_{t+1} es el punto actualizado. El algoritmo Adam es muy estable debido a que usa la estrategia tipo gradiente descendiente estocástico, además tiene mucha utilidad en estrategias de entrenamiento por lotes lo que lo hace idóneo para el análisis en el procesamiento de imágenes [5].

Otro factor importante a tener en cuenta en una red neuronal convolucional es el cruce de entropía categórica, la cual es una función que se usa para calcular la pérdida de información como se describe en eq(11). Donde $\hat{y}_i$ es el i-ésimo valor escalar en la salida del modelo, y_i es el valor objetivo correspondiente y output size es el número de valores escalables en la salida del modelo. Esta pérdida es una buena medida de cuan distinguibles son dos distribuciones de probabilidad discretas entre sí, el signo – asegura que la pérdida se reduce cuando las distribuciones se acercan entre sí. Además, la entropía cruzada categórica se adapta bien a las tareas de clasificación, por lo que es conveniente usarla en la clasificación de imágenes [6].

Implementación en Python.

Para la implementación en código del proyecto, el cual trata de clasificación de imágenes, cuyo objetivo es reconocer un número escrito a mano utilizando el mouse. Se hizo uso de la librería keras junto con sus módulos mnist, sequential, Dense, to_categorical y load_model, además se utilizó la librería Numpy, cv2, pickle y tkinter, esta última se implementó únicamente para la interfaz gráfica.

Inicialmente, se utiliza el módulo mnist para definir las imágenes de entrenamiento, las cuales están expresadas como un arreglo numpy y consta de 6.000 muestras de entrenamiento, es decir, 6.000 números en el rango de 0 a 9 escritos de diferente forma, a estos datos se adicionaron 200 imágenes escritas a mano mediante el uso del mouse; cada número tuvo una muestra de 20 imágenes, lo cual hace que haya más precisión en el modelo de la red neuronal. Las muestras de entrenamiento constan de una matriz cuadrada con dimensiones de 28x28, donde la imagen está en escala de grises. Se normalizaron los datos de entrada dividiéndolos en 255, ya que este número es el máximo valor alcanzado en cada pixel de la imagen que representa el color blanco.

Además, se usó una estrategia de entrenamiento, la cual consiste en guardar el modelo entrenado para ahorrar tiempo a la hora de hacer la predicción, el modelo es guardado con una extensión .h5 donde posteriormente se lee dicho archivo para luego hacer la respectiva predicción.

B. Abreviaturas y acrónimos

Adagrad: Adaptive Gradient Algorithm.
Adam: Adaptive Moment Estimation.
Nadam: Nesterov-accelerated Adaptive Moment Estimation.
SGD: Stochastic gradient descent.
ReLU: Rectified Linear Unit.

C. Equaciones

$$Z = X * f \quad (1)$$

$$Z[(n - f + 1)][(n - f + 1)] \quad (2)$$

$$\mathrm{Z} = W^T * \mathrm{X} + \mathrm{b} \quad (3)$$

$$f(x) = \frac{1}{(1+e^{-x})} \quad (4)$$

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (5)$$

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{k} e^{x_k}} \quad (6)$$

$$A = sigmoide(Z_1) \quad (7)$$

$$Z_2 = W^T * A + b \quad (8)$$

$$output = sigmoide(Z_2) \quad (9)$$

$$X_{t+1} = X_t - \frac{a}{\sqrt{\hat{v}_t + \varepsilon}} * \hat{p}_t \quad (10)$$

$$loss = -\sum_{i=1}^{size}{}^{output} y_i * \log(\hat{y}_i) \quad (11)$$

D. Tablas y Figuras

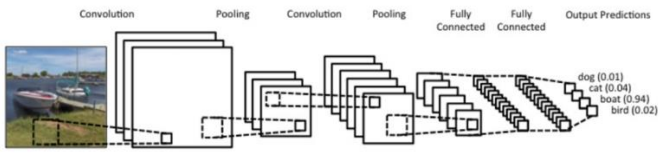

Fig 1. Representación de red neuronal convolucional. [3]

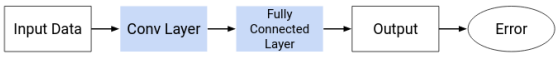

Fig 2. Arquitectura de red neuronal convolucional (CNN) [4]

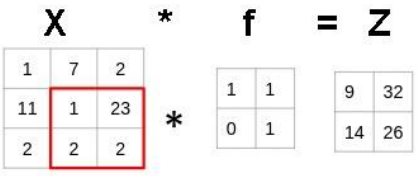

Fig 3. Convolución de la imagen de entrada X con el filtro f [4].

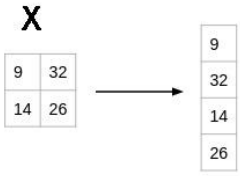

Fig 4. Redimensión imagen de entrada X a vector columna [4].

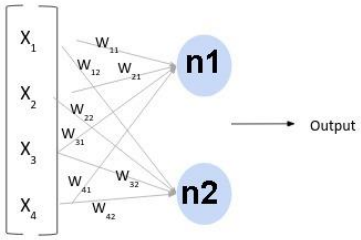

Fig 5. Estructura de una red neuronal con sus respectivos pesos [4].

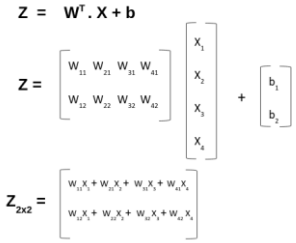

Fig 6. Transformación lineal con dos neuronas [4].

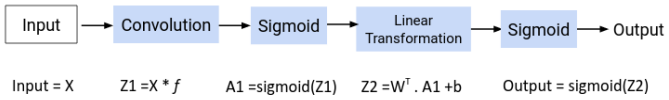

Fig 7. Arquitectura de las capas ocultas de una red neuronal convolucional [4].

III. Conclusiones

- La precisión en la predicción depende de la función de activación que se usa en cada capa junto con el número de épocas que se usa en la estructura de la red neuronal.
- Es necesario adicionar muestras a la base de datos de keras, debido a que las imágenes generadas mediante la escritura manual de los dígitos difieren en los pixeles presentes en los bordes de cada dígito.
- El tiempo de ejecución usado durante el entrenamiento depende del tipo de optimizador que se utilice, ya que este busca la dirección en la que se encuentra la solución de manera óptima.

REFERENCIAS

[1] Jaime Durán Suárez. Redes neuronales convolucionales en R, reconocimiento de caracteres escritos a mano, Sevilla, 2017.

[2] BODERO Elba M., LOPEZ Milton P. y otros. Revista Espacios, Google Colaboratory como alternativa para el procesamiento de una red neuronal convolucional, marzo, 2020.

[3] Redes neuronales convolucioanles en profundidad. Jesús, diciembre, 2018, tomado de https://datasmarts.net/es/redes-neuronales-convolucionales-en-profundidad/

[4] Demystifying the Mathematics Behind Convolutional Neural Networks (CNNs), Aishwarya Singh, febrero, 2020, tomado de https://www.analyticsvidhya.com/blog/2020/02/mathematics-behind-convolutional-neural-network/

[5] Descenso de gradiente y variaciones sobre el tema, Mariano Rivera, agosto, 2018, tomado de http://personal.cimat.mx:8181/~mrivera/cursos/optimizacion/descenso_grad_estocastico/descenso_grad_estocastico.html

[6] Categorical crossentropy, Peltarion, tomado de https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy