

Investigación y Aplicación de la Red Neuronal Artificial: LeNet

John Sebastián Galindo Hernández, Miguel Ángel Moreno Beltrán

Abstract—The objective of this lab is to research the basics of deep learning and convolutional neural networks (CNN) with a focus on the LeNet neural network and its actual application. The LeNet network was proposed by Yann LeCun in 1998 and has been used in applications of handwritten digit recognition, such as the MNIST dataset. In this lab, a database of images known as SHVN is used, which contains street houses numbers images. The results has the main porpose of evaluate the strengths and limitations of the LeNet network in its prediction capacity when a number is read by camera.

Resumen—El objetivo de este laboratorio es investigar las bases del deep learning y las redes neuronales convolucionales (CNN) con un enfoque en la red neuronal LeNet y su aplicación en la actualidad. La red LeNet fue propuesta por Yann LeCun en 1998 y ha sido utilizada en aplicaciones de reconocimiento de dígitos escritos a mano, como el conjunto de datos MNIST. En este laboratorio, se usa una base de datos de imágenes conocida como SHVN, que contiene imágenes de los números que identifican las casas. Los resultados tienen como objetivo evaluar las fortalezas y limitaciones de la red LeNet en su capacidad de predicción cuando se lee un número por medio de cámara.

Index Terms—backpropagation, cnn, redes neuronales, tratamiento de imagen, deep learning.

I. INTRODUCCIÓN

EN este laboratorio, se aborda el estudio de la red neuronal convolucional LeNet, una arquitectura pionera en el área del deep learning y las CNN. El enfoque del laboratorio es conocer de forma teórica todo lo relacionado a LeNet y evaluar cómo esta red funciona para la clasificación y predicción diferentes tipos de entradas, como números, letras y algunos símbolos contenidos en el código ASCII. A lo largo de la aplicación, se evaluará la diferencia entre el entrenamiento de la red LeNet 5 y su rendimiento con el uso de la función de activación **Tangente Hiperbólica** y el optimizador de gradiente descendiente en comparación con el entrenamiento usando **ReLU** para capas ocultas y el método Adam para la optimización de la red. Por otro lado, se evaluará el rendimiento de la red LeNet 5 en la predicción de números escritos a mano y números capturados por cámara.

II. FUNDAMENTOS TEÓRICOS

Para entender el funcionamiento de las redes neuronales artificiales, es necesario conocer los conceptos del deep learning y las redes neuronales convolucionales. Además, es importante conocer la historia de la red LeNet y su contribución al campo de la inteligencia artificial. Con base en lo anterior, se presenta cada una de las versiones que ha tenido la red LeNet, sus arquitecturas y los propósitos con las que fueron creadas, así como también el tipo de imágenes que se utilizan

para entrenarlas y sus rendimientos en comparación con otros métodos de clasificación usados en la predicción de imágenes.

A. Deep Learning

El término deep learning se refiere a un subconjunto del machine learning en donde se usan redes neuronales multicapa que permiten simular el comportamiento del cerebro humano. En el deep learning se usan tres o más capas neuronales, siendo esta su gran diferencia con los modelos de machine learning tradicionales. Otra de las características del deep learning es que aunque pueden trabajar con datos estructurados y etiquetados, también se pueden usar datos no estructurados como imágenes, audio y texto. En el deep learning, las redes neuronales permiten extraer características, rasgos y relaciones para obtener resultados exitosos con gran precisión.

El funcionamiento del deep learning se basa en que a través de una cantidad de capas con conexiones entre ellas, se pasan los datos de entrada y se obtiene una salida al realizar un proceso de cálculo ponderación y activación. En cada capa, se realiza una operación matemática que permite obtener un resultado, el cual se pasa a la siguiente capa para realizar el mismo proceso. Cada capa refina y optimiza la predicción. Estos cálculos son denominados propagación hacia adelante. Una vez que se obtiene la salida, se compara con la salida real y mediante el proceso llamado retropropagación, se utilizan algoritmos como el gradiente descendiente o el algoritmo Adam para ajustar los pesos y sesgos a modo de mejorar la precisión de la red.

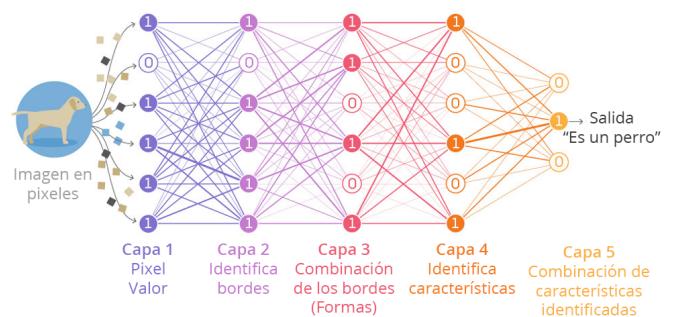


Figura 1. Proceso de entrenamiento de una red deep learning [1]

Entre los tipos de modelos de deep learning se encuentran 6 principales, los cuales se explican de manera breve a excepción de las redes neuronales convolucionales, las cuales son el enfoque principal del presente trabajo de investigación.

- **Redes Neuronales Convolucionales (CNN)**: utilizadas principalmente en el procesamiento de imágenes y vi-

sión por computadora, conocidas por su capacidad para capturar características espaciales.

- **Redes Neuronales Recurrentes (RNN)**: diseñadas para trabajar con datos secuenciales, como series temporales o lenguaje natural.
- **Redes Generativas Adversarias (GAN)**: compuestas de dos redes que compiten entre sí para generar datos sintéticos que se asemejan a datos reales, ampliamente utilizadas en generación de imágenes y contenido.
- **Autoencoders**: redes que aprenden una representación comprimida de los datos de entrada y son útiles en tareas de reducción de dimensionalidad y compresión.
- **Modelos de difusión**: Son modelos generativos entrenados con difusión directa e inversa para adicionar y eliminar ruido de las imágenes. estos modelos generan datos en base a los datos de entrenamiento y añaden ruido a los datos generados para aprender un proceso de eliminación de ruido.
- **Modelos de transformadores**: Estos modelos usan una estructura de codificador y decodificador para procesar texto. El codificador transforma el texto no procesado a representaciones llamadas incrustaciones. Despues de esto, el decodificador usa esas incrustaciones resultantes junto con palabras generadas previamente para generar texto que prediga la siguiente palabra en una oración de manera secuencial. [2]

B. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal que se utiliza principalmente en el procesamiento de imágenes y visión, aunque también son usadas en voz o audio. Estas redes neuronales se constituyen de varias capas las cuales pueden pertenecer a tres tipos: capa convolucional, capa de agrupación y capa completamente conectada. Para entender el funcionamiento de las redes neuronales convolucionales, es necesario comprender el porque una imagen puede ser clasificada en diferentes categorías; esto se debe a que las imágenes estan compuestas por pixeles, los cuales son los elementos basicos de una imagen. Cada pixel tiene un valor de intensidad que puede variar entre 0 y 255, donde 0 representa el menor nivel de intensidad y 255 el mayor nivel de intensidad.

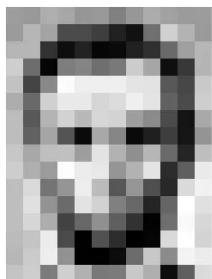


Figura 2. Imagen en escala de grises junto con sus valores por píxel [3]

157	153	174	168	150	162	129	153	172	163	155	166
155	182	160	74	75	62	39	17	158	210	180	154
180	190	50	54	54	6	10	35	48	100	159	167
206	109	8	133	131	111	120	204	156	15	56	180
194	68	137	281	237	239	239	228	227	87	71	207
172	106	207	230	239	214	230	239	228	88	74	206
188	88	178	200	185	216	211	188	179	75	20	160
189	97	186	54	10	168	154	11	31	62	22	148
199	148	191	192	158	227	178	142	106	36	190	190
205	174	186	202	236	231	149	178	228	43	95	234
190	216	179	180	216	187	83	150	78	38	218	241
190	224	147	186	227	210	127	125	36	181	258	224
190	214	172	56	103	143	91	50	2	109	249	215
187	195	235	75	1	81	47	0	6	217	255	211
183	202	237	145	6	0	12	109	209	138	243	236
195	206	125	207	177	121	122	200	175	18	96	218

Por otra parte, las imágenes pueden tener más de un canal de color, como RGB (Red, Green, Blue), en este caso, cada canal de color es una matriz de pixeles que representa un color en particular.

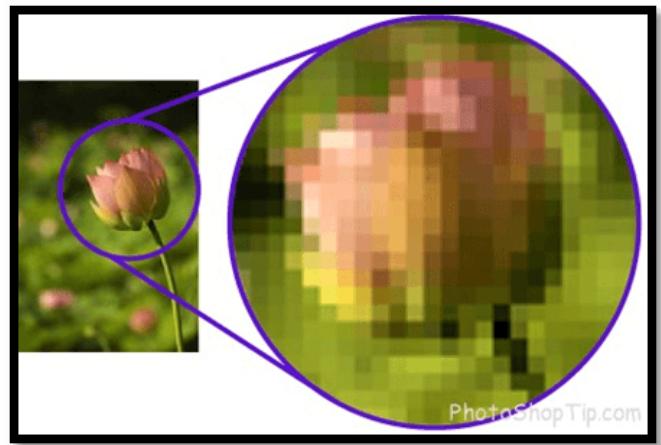


Figura 3. Píxeles en una imagen [4]

En adición, algunos conceptos básicos son definidos para asegurar la correcta comprensión de las redes neuronales convolucionales:

- **Neurona**: Es la unidad básica de una red neuronal, la cual recibe una entrada y produce una salida al realizar una suma ponderada de sus conexiones a la que se le aplica una función de activación.
- **Función de activación**: Es una función matemática que se aplica a la salida de una neurona para determinar si esta se activa o no. Algunas de las funciones de activación más comunes son la función sigmoide, la función ReLU y la función tangente hiperbólica.
- **Capa de entrada**: Es la capa que recibe los datos de entrada, en esta, cada neurona corresponde a una de las características de entrada, como los pixeles de una imagen. Por ejemplo, si se tiene una imagen de 32x32 pixeles, la capa de entrada tendrá 1024 neuronas.
- **Capa oculta**: Son las posibles capas entre la capa de entrada y la capa de salida, las salidas de las neuronas de una capa oculta se convierten en las entradas de la siguiente capa.
- **Capa de salida**: Es la capa que produce la salida final de la red neuronal, en el caso de una red de clasificación, la capa de salida tendrá tantas neuronas como clases se desean clasificar. [5]

En cuanto a las funciones de activación, se explica el funcionamiento de cada una de ellas:

- **Función Sigmóide**: En algunos ejemplos es usada pero no es habitual, la función sigmoidal se utiliza en la capa de salida de una red neuronal para problemas de clasificación binaria, ya que su salida se encuentra entre 0 y 1. Esto permite interpretar la salida como una probabilidad de pertenencia a una clase específica. Sin embargo, su uso ha disminuido en favor de otras funciones de activación, como ReLU, debido a problemas como el "desvanecimiento del gradiente" la saturación en valores extremos.

A pesar de esto, la sigmoide sigue siendo fundamental para entender cómo las redes neuronales pueden modelar relaciones no lineales y aprender patrones complejos [6].

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

Sigmoid Activation Function

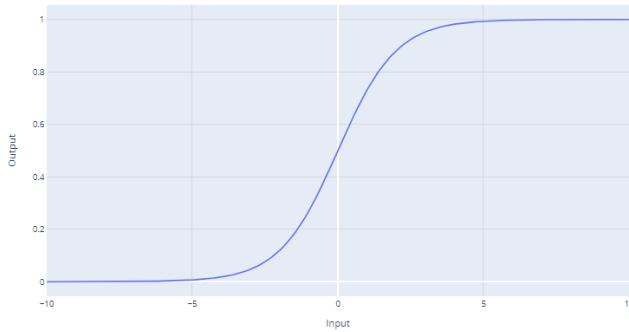


Figura 4. Función Sigmoida [7]

- **Función Tangente Hiperbólica:** La función de activación **tanh** (tangente hiperbólica) transforma valores reales en un rango de -1 a 1, siendo útil en redes neuronales para centrar los datos alrededor de cero. Su derivada es $1 - \tanh^2(x)$, lo que puede causar saturación y desvanecimiento del gradiente en redes profundas. A pesar de esto, es popular en capas ocultas por su capacidad de manejar tanto valores positivos como negativos. [8]

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2)$$

Hyperbolic Tangent Activation Function

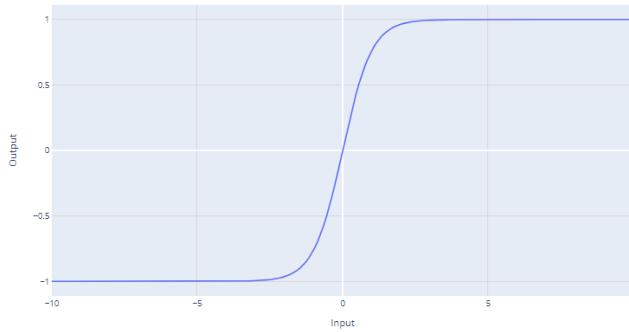


Figura 5. Función Tangente Hiperbólica [9]

- **Función ReLU (Rectified Linear Unit):** La función de activación ReLU (Unidad Lineal Rectificada) es ampliamente utilizada en redes neuronales por su simplicidad y efectividad, definida como:

$$f(x) = \max(0, x)$$

ReLU retorna 0 para valores negativos y el valor original para valores positivos, permitiendo que el gradiente

pase sin alteración en la retropropagación y evitando el problema del gradiente que se desvanece. Aunque es lineal para entradas positivas, su no linealidad en $x = 0$ le permite captar patrones complejos. Además, su bajo costo computacional y tendencia a activaciones esparsas favorecen la eficiencia en redes profundas. [8]

ReLU Activation Function

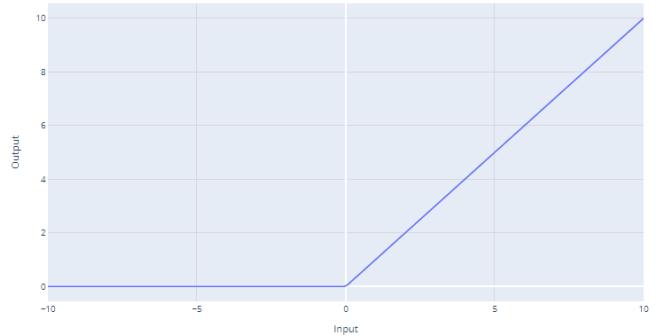


Figura 6. Función ReLU [10]

- **Función Softmax:** La función de activación Softmax es útil para clasificación multiclase, convirtiendo un vector de entradas x_1, x_2, \dots, x_C en una distribución de probabilidad:

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Cada salida representa la probabilidad de pertenencia a una clase, sumando a uno. Softmax amplifica las diferencias entre entradas, haciendo que el valor más alto domine, y se emplea comúnmente en la capa final de redes neuronales para tareas de clasificación multiclase, proporcionando interpretaciones probabilísticas de confianza en las predicciones. [8]

Softmax Activation Function

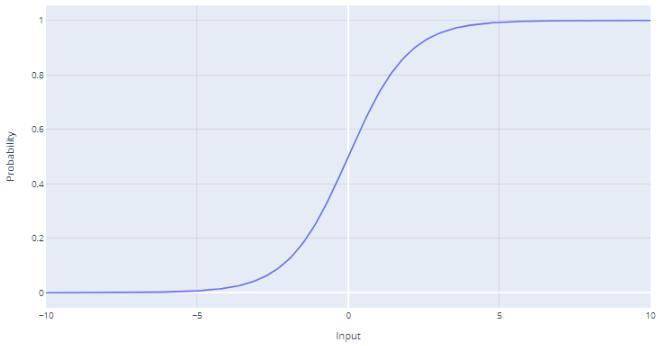


Figura 7. Función Softmax [11]

Una vez que se tienen claros los conceptos básicos, en las CNN se tienen tres tipos de capas, las cuales fueron descritas anteriormente:

- 1) **Capa Convolucional:** Esta capa es la parte fundamental de las redes neuronales convolucionales, en esta capa son

necesarios los elementos normales como las entradas, lo que la hace diferente es su necesidad de un filtro y un mapa de características. El filtro, también conocido como Kernel, es una matriz de pesos (números) que se aplica a la imagen de entrada, este kernel se desliza través de cada píxel para extraer características tales como bordes, texturas, formas, entre otros. La imagen resultante se compone de un producto punto entre el kernel y cada pixel de la imagen. Los Kernel pueden ser de diferentes tamaños y pueden ser predefinidos o aprendidos durante el entrenamiento de la red. Un ejemplo de un Kernel habitualmente usado es el Kernel de Sobel, el cual se utiliza para detectar bordes en una imagen.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Figura 8. Kernel de Sobel Horizontal y Vertical [12]

Otro de los aspectos clave es que el tamaño del resultado se ve afectado por los siguientes hiperparametros:

- **Cantidad de filtros:** La cantidad de filtros afecta a la profundidad de la salida, esto porque cada filtro produce un mapa de características. Por ejemplo, si se tienen 6 filtros y se recibe una imagen con un solo canal de color, la salida tendrá 6 canales de color, pero si se recibe una imagen con 3 canales de color como RGB, la salida tendrá 18 canales de color (18 mapas de características).
- **Stride:** Es el número de píxeles que se desplaza el filtro en cada paso.
- **Padding:** Es la cantidad de píxeles que se añaden alrededor de la imagen, generalmente se usa cuando el filtro es más grande que la imagen, el borde que se le añade a la imagen es con valores de 0.
- **Tamaño del filtro:** Es el tamaño del filtro que se desliza sobre la imagen, generalmente se usan filtros de 3x3 o 5x5.

El calculo final para el tamaño que tendra un mapa de características despues de aplicar un filtro a una imagen es el siguiente:

$$\text{Salida} = \frac{N - K + 2P}{S} + 1 \quad (3)$$

Donde:

- **N:** Tamaño de la imagen de entrada.
- **K:** Tamaño del filtro.
- **P:** Padding.
- **S:** Stride.
- **Salida:** Tamaño de la imagen de salida.

Un ejemplo claro del calculo realizado con un filtro de 3x3 y una imagen de 5x5 es el siguiente:

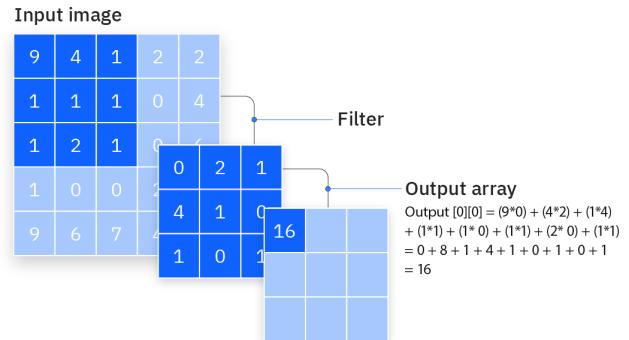


Figura 9. Capa convolucional aplicada a una imagen [13]

Cabe aclarar que puede haber más de una capa convolucional en una red, esto convierte la estructura de la CNN en jerárquica, donde cada capa que sigue a la anterior, aprende características más complejas.

2) **Capa de Agrupación:** La capa de agrupación (Pooling) es una capa que se utiliza para reducir las dimensiones (Ancho y Alto) de la imagen, para disminuir el número de parámetros y el costo computacional. La agrupación aplica una función de agregación a un conjunto de píxeles, como el promedio o el máximo; Esta capa aunque pierde información, ayuda a mejorar la generalización de la red y su eficiencia. La capa de agrupación puede ser de los siguientes tipos [14]:

- **Max Pooling:** Se selecciona el valor máximo de un conjunto de píxeles, por lo tanto la salida después de la capa de max pooling será el mapa de características con los valores más altos.

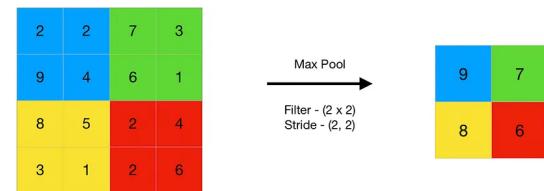


Figura 10. Max Pooling aplicado a un mapa de características [15]

- **Average Pooling:** Se calcula el promedio de un conjunto de píxeles en el mapa de características, por lo tanto la salida después de la capa de average pooling será el mapa de características con los valores promedio.

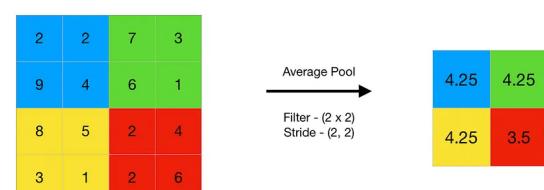


Figura 11. Average Pooling aplicado a un mapa de características [16]

Además de estos tipos, también hay agrupamiento global, el cual calcula el promedio o el máximo de cada canal de características, en lugar de hacerlo por regiones; Esto da como salida un solo valor por canal de características.

3) *Capa Completamente Conectada*: Normalmente, suele haber una capa entre la capa de agrupación y la capa completamente conectada, esta capa se conoce como capa de aplanamiento (Flatten), la cual convierte la matriz de características en un vector unidimensional, para que pueda ser procesado por la capa completamente conectada.

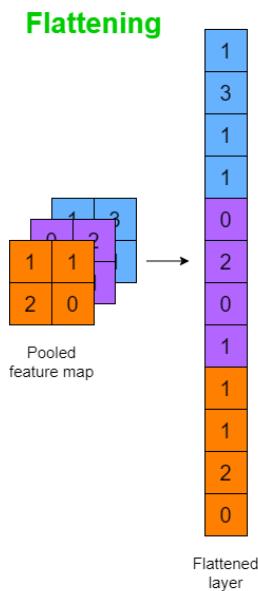


Figura 12. Capa de aplanamiento aplicada a tres mapas de características [17]

Por último, las capas completamente conectadas son las capas de una RNA tradicional, donde cada neurona de la capa está conectada con todas las neuronas de la capa anterior. Estas capas se utilizan para clasificar las características extraídas por las capas convolucionales y de agrupación. La capa completamente conectada se encarga de realizar la clasificación final de la red, en donde se obtiene la probabilidad de que una imagen pertenezca a una clase en particular mediante la función de activación softmax.

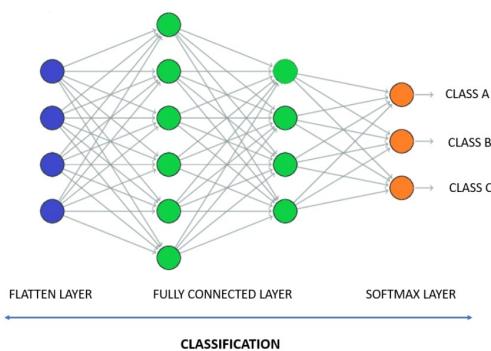


Figura 13. Capa completamente conectada [18]

C. LeNet

1) *Historia*: La historia de LeNet refleja un progreso innovador en el desarrollo de redes neuronales convolucionales aplicadas a la identificación de patrones visuales. En 1988, Yann LeCun se unió al Departamento de Investigación de Sistemas Adaptativos en AT&T Bell Labs, dirigido por Lawrence D. Jackel. Desde entonces, LeCun y su equipo comenzaron a trabajar en modelos que pudieran reconocer caracteres manuscritos de manera eficiente, en particular códigos postales.

En 1989, publicaron la primera versión de una red neuronal convolucional (LeNet-1) diseñada específicamente para leer dígitos manuscritos. Esta versión utilizaba núcleos de convolución fijos y se entrenaba mediante el algoritmo de retropropagación, lo cual era una innovación importante en el campo. Este primer modelo mostró que la combinación de redes convolucionales y retropropagación podía aprender a generalizar características visuales relevantes con una alta precisión, incluso en tareas complejas como el reconocimiento de dígitos manuscritos proporcionados por el Servicio Postal de los Estados Unidos. En este año también, LeCun demostró que la restricción del número de parámetros libres mejoraba significativamente la capacidad de generalización del modelo.

En 1990, se publicaron más investigaciones sobre la aplicación de redes de retropropagación para reconocimiento de caracteres, logrando un margen de error bajo en la identificación de números en imágenes, con una tasa de rechazo controlada. Estos resultados alentaron investigaciones continuas, y en 1994, LeCun y su equipo desarrollaron la base de datos MNIST, diseñada como un estándar para evaluar redes de reconocimiento de dígitos. LeNet-4 se entrenó para abordar el tamaño y la complejidad de esta base de datos.

El esfuerzo culminó en 1995 con el desarrollo de LeNet-5, una arquitectura más avanzada que integraba métodos adicionales para mejorar la precisión en la clasificación de caracteres manuscritos. Este modelo no solo superó otros métodos en pruebas de referencia, sino que también fue aplicado en sistemas de lectura automática de millones de cheques bancarios al día. En 1998, LeCun y su equipo presentaron estos avances como ejemplos de aplicaciones prácticas en el reconocimiento de caracteres manuscritos.

Aunque las redes neuronales modernas han evolucionado significativamente desde LeNet, esta serie de modelos sentó las bases para el diseño de arquitecturas convolucionales complejas, marcando un hito en la historia de la inteligencia artificial y el aprendizaje profundo [19] [20].

2) *versiones*:

- **LeNet-1:** LeNet-1 fue uno de los primeros modelos de redes neuronales convolucionales diseñado por Yann LeCun y su equipo. La arquitectura estaba enfocada en reconocer dígitos manuscritos y se basaba en la aplicación de capas de convolución y capas de subsampling o pooling. Los detalles de su arquitectura son los siguientes:

Entrada: Imagen de 28x28 píxeles en escala de grises.

Capa de convolución 1: genera cuatro mapas de características de 24x24 píxeles mediante la convolución con kernels predefinidos de 5x5 píxeles.

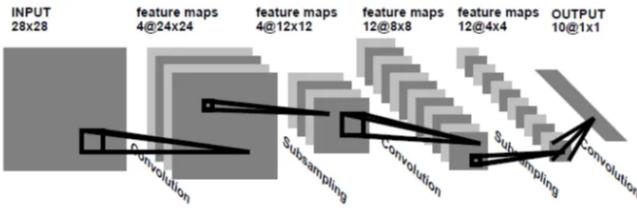


Figura 14. Arquitectura de LeNet-1 [21]

Capa Average Pooling 1: reduce la dimensión de los mapas de características a 12x12 píxeles usando una ventana de 2x2 píxeles para calcular el promedio.

Capa de convolución 2: produce 12 mapas de características de 8x8 píxeles mediante la convolución con kernels de 5x5 píxeles.

Capa Average Pooling 2: reduce la dimensión de los mapas de características a 4x4 píxeles usando una ventana de 2x2 píxeles para calcular el promedio.

Capa de Salida: capa completamente conectada con 10 neuronas para clasificar los dígitos del 0 al 9.

- **LeNet-4:**

LeNet-4 es una versión mejorada de LeNet-1, diseñada para manejar imágenes de entrada ligeramente más grandes y con más capacidad de aprendizaje. La arquitectura de LeNet-4 tiene los siguientes componentes:

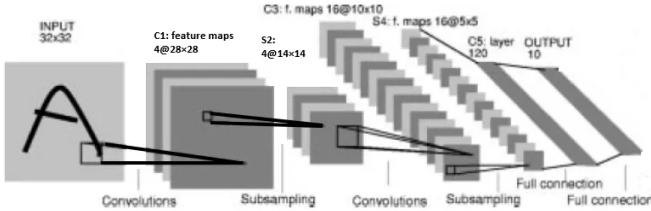


Figura 15. Arquitectura de LeNet-4 [22]

Entrada: Imagen de 32x32 píxeles en escala de grises.

Capa de convolución 1: genera cuatro mapas de características de 28x28 píxeles mediante la convolución con kernels de 5x5 píxeles.

Capa Average Pooling 1: reduce la dimensión de los mapas de características a 14x14 píxeles usando una ventana de 2x2 píxeles para calcular el promedio.

Capa de convolución 2: produce 16 mapas de características de 10x10 píxeles mediante la convolución con kernels de 5x5 píxeles.

Capa Average Pooling 2: reduce la dimensión de los mapas de características a 5x5 píxeles usando una ventana de 2x2 píxeles para calcular el promedio.

Flatten Layer: convierte los mapas de características en un vector unidimensional.

Capa completamente conectada 1: 120 neuronas con función de activación sigmoidal o tangente hiperbólica.

Capa de Salida: 10 neuronas con función de activación softmax para clasificar los dígitos del 0 al 9.

- **LeNet-5:**

LeNet-5 es la versión más popular de la arquitectura

LeNet, introducida en 1995. Es similar a LeNet-4, pero incluye algunos ajustes adicionales en la cantidad de neuronas y conexiones. Su estructura es la siguiente:

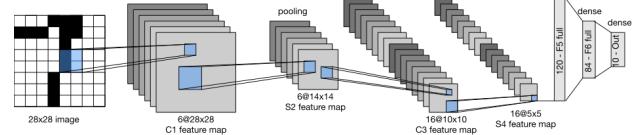


Figura 16. Arquitectura de LeNet-5 [23]

Entrada: Imagen de 32x32 píxeles en escala de grises.

Capa de convolución 1: genera seis mapas de características de 28x28 píxeles mediante la convolución con kernels de 5x5 píxeles.

Capa Average Pooling 1: reduce la dimensión de los mapas de características a 14x14 píxeles usando una ventana de 2x2 píxeles para calcular el promedio.

Capa de convolución 2: produce 16 mapas de características de 10x10 píxeles mediante la convolución con kernels de 5x5 píxeles.

Capa Average Pooling 2: reduce la dimensión de los mapas de características a 5x5 píxeles usando una ventana de 2x2 píxeles para calcular el promedio.

Flatten Layer: convierte los mapas de características en un vector unidimensional.

Capa completamente conectada 1: 120 neuronas con función de activación sigmoidal o tangente hiperbólica.

Capa completamente conectada 2: 84 neuronas con función de activación sigmoidal o tangente hiperbólica.

Capa de Salida: 10 neuronas con función de activación softmax para clasificar los dígitos del 0 al 9.

- **Boosted LeNet-4:**

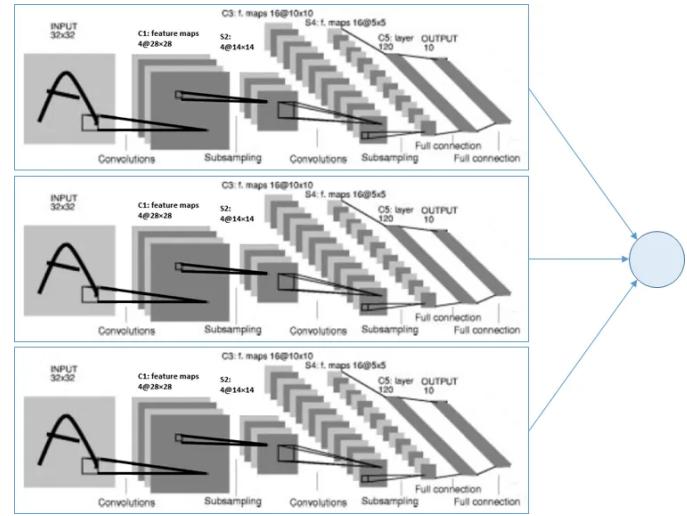


Figura 17. Arquitectura de Boosted LeNet-4 [24]

En una versión mejorada llamada Boosted LeNet-4, se implementó una técnica de combinación de múltiples

redes para aumentar la precisión de LeNet-4. Esta técnica de boosting combinaba los resultados de tres redes LeNet-4; se sumaban las salidas y el valor más alto era la clase predicha. Además, si una red obtenía una salida con alta confianza, las demás redes no se usaban. Con esta modificación, Boosted LeNet-4 redujo la tasa de error a 0.7 %, mejorando incluso a LeNet-5.

3) Aplicaciones: La red neuronal convolucional LeNet ha demostrado ser una arquitectura fundamental que trasciende su propósito inicial de reconocimiento de dígitos manuscritos tal como se ve presente en el desarrollo realizado por Yann LeCun en su artículo sobre LeNet, en aquella ocasión la implementación logró una precisión de clasificación de aproximadamente el 99,2 %, aunque el documento original es largo y trata varios temas, cabe recalcar la comparación con modelos de clasificación del momento dando un porcentaje de error bastante más bajo con respecto a los otros modelos, entre los modelos contra los cuales fue comparado están: K-NN, Support Vector Machine y diferentes combinaciones de capas de neuronas.

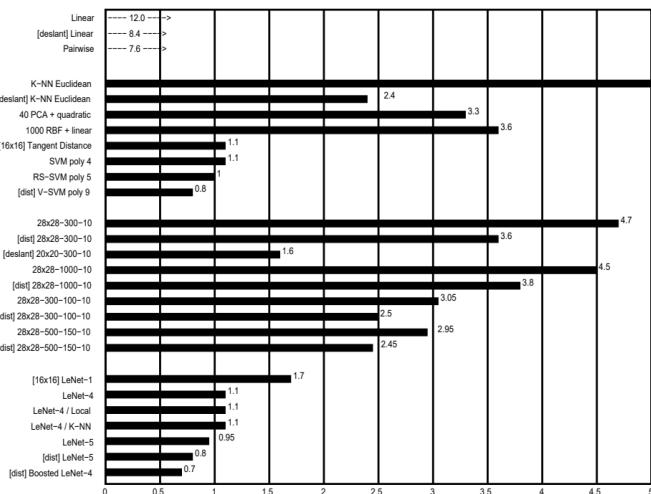


Figura 18. Comparación de LeNet con otros modelos de clasificación [25]

Sus aplicaciones se han diversificado significativamente en diversos campos tecnológicos y científicos. En el sector de procesamiento documental, esta arquitectura facilita la automatización de tareas de gestión y clasificación de documentos, mientras que en el ámbito médico contribuye al análisis de imágenes diagnósticas para la identificación de anomalías, un claro ejemplo de esto es el artículo llamado A Modified LeNet CNN for Breast Cancer Diagnosis in Ultrasound Images [26] donde se utiliza una versión modificada de LeNet para la detección de cáncer de mama en imágenes de ultrasonido, este modelo se planteó como un clasificador de tumores benignos y malignos, obteniendo una precisión del 89.91 % en el reconocimiento de imagen. El sistema también ha revolucionado los procesos de autenticación biométrica mediante el análisis de firmas y huellas dactilares, fortaleciendo los protocolos de seguridad. En el contexto del análisis de video en tiempo real, los principios de LeNet han sido instrumentales para el desarrollo de sistemas de vigilancia y reconocimiento facial,

En este campo, se tiene como ejemplo práctico el artículo online publicado por Andrii Gozholovskyi llamado Classification of Traffic Signs with LeNet-5 CNN [27] donde se usan los datos de las señales de tráfico alemanas para entrenar una red LeNet-5 que permite clasificarlas, este artículo comprende un tutorial paso a paso de cómo implementar la red junto con sus datos, evaluación y graficación de resultados y métricas. Adicionalmente, su implementación en el reconocimiento de caracteres manuscritos ha facilitado la digitalización de documentos históricos y el procesamiento automatizado de formularios en diversos sectores, incluyendo servicios postales y bancarios. La versatilidad de esta arquitectura ha permitido su adaptación para la clasificación de objetos en entornos industriales y comerciales, demostrando su valor en sistemas de control de calidad y gestión de inventarios [19].

4) Limitaciones: Con respecto a las limitaciones de la red LeNet, se pueden mencionar algunas de las siguientes:

- Capacidad de generalización:** Al ser una arquitectura relativamente simple, LeNet puede tener una limitación para generalizar tareas complejas, esto debido a su pequeño tamaño y cantidad de capas.
- Tamaño de entrada:** Se restringe su aplicabilidad a imágenes de tamaño fijo (32x32), lo cual es ineficaz para aplicaciones del mundo real donde las imágenes varían en tamaño.
- Falta de escalabilidad:** Puede ser un modelo que se queda corto en adaptaciones a datos modernos y demandas de deep learning. [20]

III. METODOLOGÍA

Esta sección describe la metodología utilizada para la implementación de la red neuronal convolucional LeNet-5. Se describen los pasos seguidos para la implementación de la red, la obtención de los datos, el preprocesamiento de los datos, la implementación de la red y la evaluación de la red. Esta red LeNet se enfocó en la clasificación de números, con una funcionalidad en tiempo real para la predicción de números capturados por cámara y fue implementada en Python con la librería TensorFlow.

A. Obtención de los datos

Los datos usados fueron obtenidos de la base de datos Housenumbers (SVHN) que contiene imágenes de números de casas. La base de datos contiene 73257 imágenes de entrenamiento y 26032 imágenes de prueba, pero se usaron los 600,000 datos extra para el entrenamiento de la red, quedando un total de 673,257 imágenes de entrenamiento y 26032 imágenes de prueba. Esta base de datos contiene un formato de imágenes de 32x32 pixeles, con 3 canales de color y se escogió para la implementación de la red LeNet-5 principalmente para adecuarla a un entorno más realista.

Estos datos están disponibles en la página web de SVHN [28] en donde se da el conjunto de datos en formato .mat, por este motivo, la lectura de los datos se usó con la librería `spicy.io` la cual permite leer el formato .mat mediante la implementación de las siguientes dos funciones:

```

def initialize_svhn_info():
    global train_images, train_labels,
        test_images, test_labels
    # Cargar los datos SVHN
    train_images, train_labels =
        load_svhn_data(get_resource_path('Data
            /train_32x32.mat'))
    test_images, test_labels = load_svhn_data(
        get_resource_path('Data/test_32x32.mat
            '))
    extra_images, extra_labels =
        load_svhn_data(get_resource_path('Data
            /extra_32x32.mat'))

    # Concatenar los conjuntos de
        # entrenamiento y extra
    train_images = np.concatenate([
        train_images, extra_images], axis=0)
    train_labels = np.concatenate([
        train_labels, extra_labels], axis=0)

    # Convertir las etiquetas a formato categó
        # rico
    train_labels = to_categorical(train_labels
        )
    test_labels = to_categorical(test_labels)

def load_svhn_data(mat_file_path):
    # Cargar el archivo .mat
    svhn_data = scipy.io.loadmat(mat_file_path
        )
    images = svhn_data['X'] # Contiene las im
        ágenes
    labels = svhn_data['y'] # Contiene las
        # etiquetas

    # Cambiar la etiqueta 10 a 0 (en SVHN, 10
        # representa el dígito 0)
    labels[labels == 10] = 0

    # Convertir imágenes de (32, 32, 3, N) a (
        N, 32, 32, 3)
    images = np.moveaxis(images, -1, 0)

    # Convertir a escala de grises y
        # normalizar
    images_gray = np.array([cv2.cvtColor(img,
        cv2.COLOR_RGB2GRAY) / 255.0 for img in
        images])

    return images_gray, labels

```

En las función llamada `initialize_svhn_info` se accede a las rutas tanto de los conjuntos de entrenamiento, pruebas y los datos extras, todos los archivos se ubicaron en una carpeta llamada **Data**, para cargar los datos se usa la segunda función, en donde mediante la función `loadmat` se carga el archivo en formato `.mat` pasandole una ruta especificada, una vez se lee la información contenida en los archivo , se separan las imagenes y se guardan en una variable (`X`) y las etiquetas de las imagenes en otra variable (`y`), después de esto, se cambia la etiqueta **10** a **0** y se convierten a un formato adecuado con la función `moveaxis` para cambiar la orientación de la matriz que representa la imagen a una valida para la manipulación en la red LeNet. Una vez que el proceso se realiza, se combinan el conjunto de datos de

entrenamiento y el extra en uno solo con la función de numpy para concatenar; Por otra parte, en la finalización del proceso se convierten las etiquetas de indentificación en una etiqueta valida para Softmax, como ejemplo, para el número **1** se tendría una etiqueta de las siguiente manera: [0,1,0,0,0,0,0,0,0] en donde 1 representaría que esa posición tiene un 100 % de probabilidad.

Algunos ejemplos de las imágenes de entrada son los siguientes:



Figura 19. Ejemplo de imágenes del conjunto de datos SHVN [28]

B. Preprocesamiento de los datos

Para el preprocesamiento de los datos, se realizó una conversion a escala de grises de las imágenes y se normalizaron los datos, esto se hizo por medio de la librería OpenCV en Python con el siguiente código que se encuentra dentro de la anterior función para el cargue de datos.

```

np.array([cv2.cvtColor(img, cv2.
    COLOR_RGB2GRAY) / 255.0 for img in
    images])

```

este código permite hacer una matriz con la cantidad de imágenes y en esa matriz se agregan los datos pasados a escala de grises con la función `cvtColor` especificando la imagen y el formato nuevo de color en los parametros, por último, se usa la division entre 255 para cada imagen y así normalizarla.



Figura 20. Ejemplo de imágenes del conjunto de datos SHVN en escala de grises

C. Implementación de la red LeNet-5

Una vez se ha realizado el tratamiento de las imágenes se debe inicializar la red LeNet, para la construcción de esta, se usaron las funcionalidades de **Keras** que se encuentran en la librería de **TensorFlow**.

Para implementar la arquitectura en código python se requiere de las siguientes líneas de código:

```
# Definir la arquitectura de LeNet-5
model = models.Sequential()
model.add(keras.Input(shape=(32, 32, 1)))
model.add(layers.Conv2D(6, (5, 5),
                      activation='relu'))
model.add(layers.AveragePooling2D(
    pool_size=(2, 2)))
model.add(layers.Conv2D(16, (5, 5),
                      activation='relu'))
model.add(layers.AveragePooling2D(
    pool_size=(2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(120, activation='relu'))
model.add(layers.Dense(84, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
# Compilar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Aquí hay varias funciones clave:

- **models.Sequential()**: Crea una instancia del modelo secuencial de Keras. Un modelo secuencial es una pila de capas donde la salida de una capa se convierte en la entrada de la siguiente capa.
- **add(layer)**: Agrega una capa al modelo.
- **keras.Input(shape=(32, 32, 1))**: Define la forma de entrada de la red. La capa de entrada especifica la forma de los datos de entrada que alimentarán al modelo. En este caso, los datos de entrada tienen una forma de (32, 32, 1), lo que significa que cada imagen tiene una dimensión de 32x32 píxeles y un solo canal (escala de grises).

- **layers.Conv2D(6, (5, 5), activation='relu')**: Agrega una capa convolucional 2D al modelo. La capa de convolución aplicará 6 filtros de 5x5 píxeles a las imágenes de entrada. Cada filtro aprenderá a detectar características particulares de las imágenes. La función de activación utilizada es ReLU (Rectified Linear Unit), que ayuda a introducir no linealidad en el modelo.
- **layers.AveragePooling2D(pool_size=(2, 2))**: Agrega una capa de agrupación promedio 2D al modelo.
- **layers.Conv2D(16, (5, 5), activation='relu')**: Agrega otra capa convolucional 2D al modelo. Esta capa aplicará 16 filtros de 5x5 píxeles a las imágenes de entrada.
- **layers.Flatten()**: Agrega una capa de aplanado al modelo.
- **layers.Dense(120, activation='relu')**: Agrega una capa densa al modelo que cuenta con 120 neuronas y utiliza la función de activación ReLU.
- **layers.Dense(84, activation='relu')**: Agrega otra capa densa al modelo que cuenta con 84 neuronas y utiliza la función de activación ReLU.
- **layers.Dense(10, activation='softmax')**: Agrega la capa de salida al modelo, que cuenta con 10 neuronas donde cada neurona representa un dígito del 0 al 9. La función de activación utilizada es Softmax, que convierte las salidas de las neuronas en probabilidades.
- **model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])**: Esta función compila el modelo. Se especifica el optimizador Adam el cual es un algoritmo de optimización que se utiliza para ajustar los pesos de la red durante el entrenamiento y que se encarga de minimizar la función de pérdida. La función de pérdida utilizada es la entropía cruzada categórica, que se utiliza para problemas de clasificación con más de dos clases en donde su representación de salida es de tipo one-hot, por ejemplo, [0, 1, 0, 0, 0, 0, 0, 0, 0]. Por último, se especifica la métrica de precisión para evaluar el rendimiento del modelo.

Cabe aclarar que aunque estas no sean las funciones que se usaron en la implementación de la red LeNet-5, estas funciones son las que actualmente tienen un mejor rendimiento en la implementación de redes neuronales convolucionales, sin embargo, la red fue probada tanto con las funciones en la implementación original de la red LeNet-5 como con las funciones actuales y se realizó una comparación de los resultados obtenidos.

D. Entrenamiento de la red

Para el entrenamiento de la red se usaron los datos de entrenamiento y se ajustaron los hiperparámetros de la red, uno de los parámetros importantes para el entrenamiento es la condición de parada, en este caso, se uso la precisión del modelo de prueba para detener el entrenamiento, esto se logra con la función **EarlyStopping** de Keras y se muestra su implementación en el siguiente código:

```

early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=3,
    min_delta=0.001,
    mode='max',
    verbose=1
)

```

En este código, se especifica que la métrica a monitorear es la precisión en los datos de validación, se establece un umbral mínimo de mejora entre épocas de 0.001 y se especifica que se busca maximizar la precisión, además, se establece un límite de paciencia de 3 épocas, lo que significa que si la precisión no mejora después de 3 épocas, el entrenamiento se detendrá.

Continuando con el entrenamiento, se usó la función **fit** de Keras para entrenar el modelo, a continuación se muestra el código de entrenamiento:

```

model.fit(
    train_images,
    train_labels,
    epochs=1000,
    batch_size=128,
    validation_data=(test_images,
                     test_labels),
    callbacks=[early_stopping]
)

```

En este código, se especifican los datos de entrenamiento y las etiquetas de entrenamiento, el número de épocas de entrenamiento, el tamaño del lote a 128, el lote es la cantidad de datos que se usan para calcular el gradiente y actualizar los pesos de la red, es decir, se actualizan los pesos después de procesar un lote de datos y no para cada patrón. También se especifican los datos de validación y las etiquetas de validación, las cuales servirán para evaluar el rendimiento del modelo durante el entrenamiento y la condición de parada. Por último, se especifica la función de parada temprana que se usará durante el entrenamiento.

E. Evaluación de la red

Una vez que el modelo está entrenado simplemente se usa la función **evaluate** de Keras para evaluar el rendimiento del modelo en los datos de prueba, a continuación se muestra el código de evaluación:

```

# Evaluar el modelo
test_loss, test_accuracy = model.evaluate(
    test_images, test_labels)

model_path = get_resource_path('Data/
    lenet_5_model.keras')
model.save(model_path)

```

En este código, se especifican los datos de prueba y las etiquetas de prueba, y se almacenan en las variables **test_loss** y **test_accuracy** respectivamente. La función **evaluate** devuelve la pérdida y la precisión del modelo en los datos de prueba, y al final se guarda el modelo en un archivo con extensión .keras para su posterior uso sin necesidad de volver a entrenar la red.

La implementación de los códigos fue basada en los tutoriales de las páginas [29] y [20].

IV. RESULTADOS

Para guiar toda la parte de resultados, se realizó una interfaz gráfica con el uso de la librería **CustomTkinter** que permite crear una aplicación completamente funcional de escritorio para que el usuario pueda interactuar con la red neuronal LeNet 5. La aplicación permite al usuario las siguientes funcionalidades:

- **Ver Datos de Entrenamiento:** El usuario puede visualizar los datos de entrenamiento, esta funcionalidad muestra 5 imágenes de ejemplo por cada clase de la base de datos.

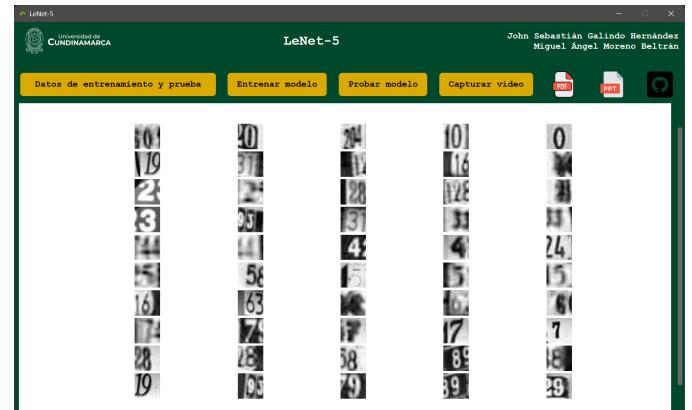


Figura 21. Datos de Entrenamiento

- **Entrenar la Red:** El usuario puede entrenar la red neuronal LeNet 5 con los datos de MNIST o SVHN, después de que el usuario seleccione la base de datos, la red carga los datos y comienza el entrenamiento, el entrenamiento se puede ver por consola y la arquitectura de la red junto con los pesos resultantes para los kernels de la primera y segunda capa convolucional.



Figura 22. Selección de la Base de Datos

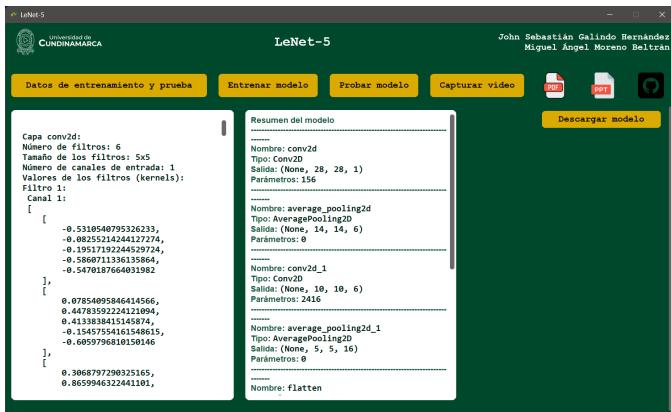


Figura 23. Entrenamiento de la Red



Figura 25. Sección de captura de Video

- Probar la Red:** El usuario puede probar la red neuronal LeNet 5 con la imagen que el seleccione desde el explorador de archivos, la red carga la imagen y la procesa para adecuarla a la red redimensionando la imagen y convirtiendola a escala de grises, después de esto la red realiza la predicción y se muestra tanto la imagen procesada como la predicción y las probabilidades de cada clase.

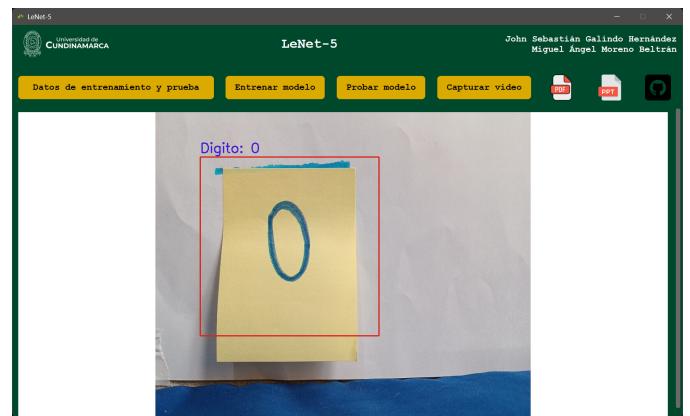


Figura 26. Captura de Video

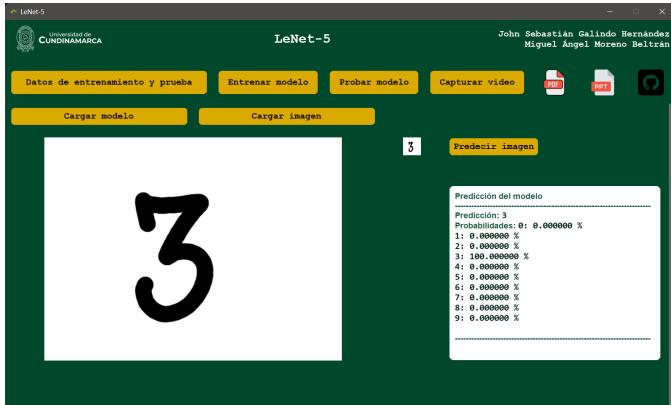


Figura 24. Sección de prueba de la Red

- Capturar video:** El usuario puede capturar un video con la cámara de su computadora, la cámara que tenga conectada o la camara por IP, ya que esta se especifica mediante un texto ingresado en la GUI. una vez se hace conexión con la camara especificada se puede capturar el video y la red realiza la predicción de los números que se encuentren en el en tiempo real.

A. Resultados del entrenamiento de la Red LeNet 5

Para el entrenamiento de la red LeNet 5 se utilizo la base de datos de MNIST y SVHN, en ambos casos se utilizó el optimizador adam y la función de activación ReLU para las capas ocultas, después de que se realizó el entrenamiento se obtuvieron los siguientes resultados:

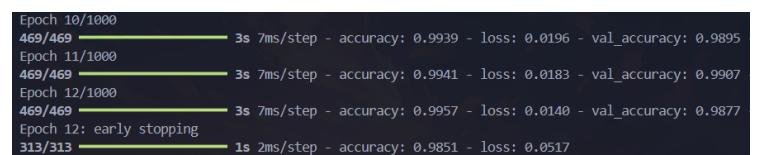


Figura 27. Resultados del entrenamiento con MNIST

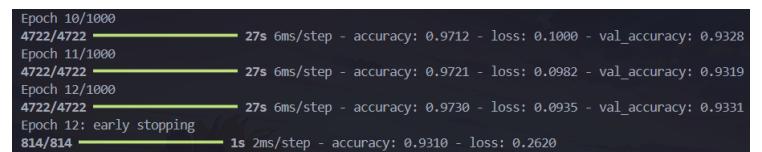


Figura 28. Resultados del entrenamiento con SVHN

Como se puede comprobar, los resultados obtenidos en el entrenamiento de la red LeNet 5 con la base de datos MNIST y SVHN son muy buenos, ya que en ambos casos se obtuvo

una precisión por encima del 90 % en el conjunto sin embargo al tener menos datos MNIST y al necesitar que las imágenes estén con la escala de grises invertida se obtiene una menor tasa de acierto en la predicción de los números que no estén lo suficientemente contrastados con el fondo. Es por este motivo, que se recomienda el uso de la base de datos SVHN para el entrenamiento de la red ya que se han realizado pruebas con la base de datos y se ha obtenido una precisión muy alta con imágenes con un ruido muy alto.

Por otra parte, se realiza la comparación entre el entrenamiento dado en la figura 28 y los mismos datos con las funciones de activación (Tangente Híperbolica) y el optimizador de gradiente descendiente los cuales se usaban originalmente en la red LeNet 5, los resultados obtenidos para la prueba original son los siguientes:

```
4722/4722 - 27s 6ms/step - accuracy: 0.9458 - loss: 0.1887 - val_accuracy: 0.8991
Epoch 17/1000
4722/4722 - 27s 6ms/step - accuracy: 0.9472 - loss: 0.1850 - val_accuracy: 0.8998
Epoch 18/1000
4722/4722 - 27s 6ms/step - accuracy: 0.9485 - loss: 0.1816 - val_accuracy: 0.9000
Epoch 19/1000
4722/4722 - 27s 6ms/step - accuracy: 0.9485 - loss: 0.1793 - val_accuracy: 0.8989
Epoch 19: early stopping
814/814 - 28 2ms/step - accuracy: 0.8949 - loss: 0.3726
```

Figura 29. Resultados del entrenamiento con SVHN

Como se puede observar, los resultados obtenidos con la función de activación ReLU y el optimizador Adam son mejores que los obtenidos con la función de activación Tangente Híperbolica y el optimizador de gradiente descendiente, ya que no solo alcanza una precisión mayor, sino que también el tiempo de entrenamiento es menor, para el entrenamiento con los hiperparametros orginales el tiempo total fue de 9 minutos y 30 segundos logrando una precisión en el conjunto de prueba de 89.5 % mientras que con los nuevos hiperparametros (ReLU y Adam) el tiempo total fue de 5 minutos y 30 segundos logrando una precisión en el conjunto de prueba de 93.1 %.

B. Resultados de la Predicción de la Red LeNet 5

Para la predicción de la red LeNet 5 se utilizo la base de datos de MNIST y SVHN, en ambos casos se tuvieron algunos errores cuando los datos no estaban tenian muchas irregularidades o cuando su posicion no era la adecuada, aún así, con los datos de MNIST se obtuvo un grande error con al probar con imágenes de numeros que tienen estilos de caligrafia. Por otro lado, con la base de datos SVHN se obtuvieron resultados muy buenos, tanto para caligrafias de computadora como para datos escritos a mano e incluso para imágenes de numeros en pinturas o con texturas muy ruidosas.

Uno de los ejemplos de un resultado presentado trabajando con el modelo entrenado con la base de datos SVHN es el siguiente:

Aún cuando hay resultados buenos, el modelo no es perfecto y se puede mejorar, por ejemplo, en la siguiente predicción se puede observar que el modelo no es capaz de predecir correctamente el número y termina distribuyendo la probabilidad entre varios números lo cual no es habitual en la red LeNet 5.

todas las imágenes de prueba fueron obtenidas del conjunto de datos Numerical Images encontrado en la pagina web de Kaggle [30].

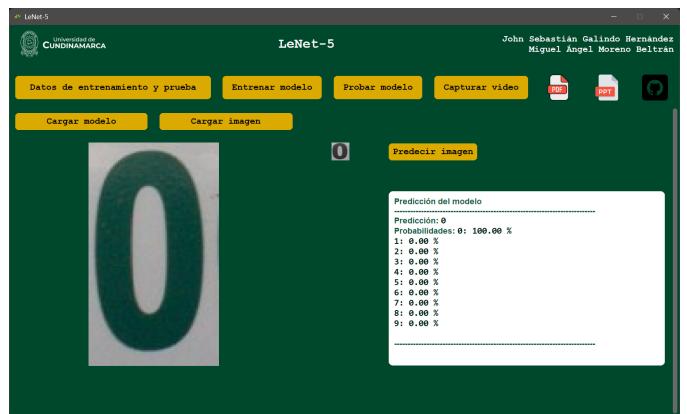


Figura 30. Predicción de la Red LeNet 5 para el número 0

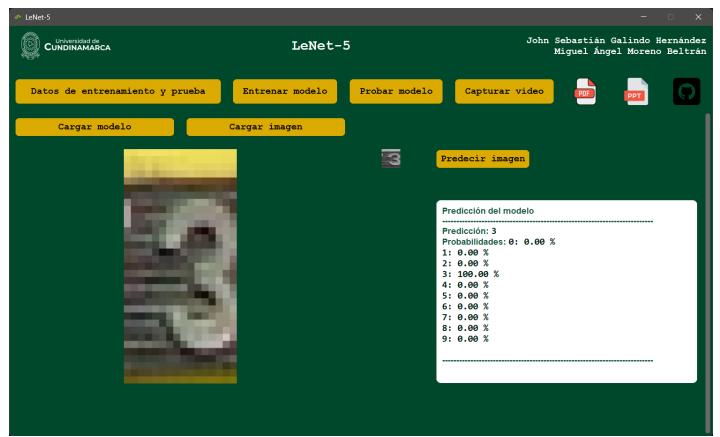


Figura 31. Predicción de la Red LeNet 5 para el número 3 con baja resolución

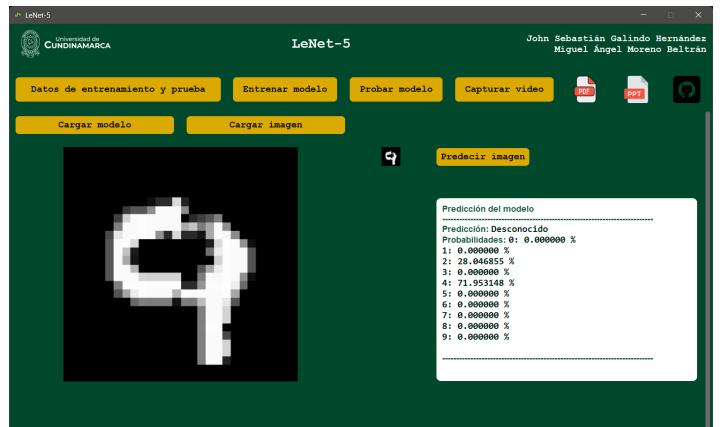


Figura 32. Predicción de la Red LeNet 5 para el número 9

C. Resultados de la Captura de Video

Para la captura de video se utilizó la cámara de un celular conectado mediante IP con una resolución de 1080p, en la captura de video se obtuvieron resultados muy buenos pero que eran muy susceptibles a la posición del número en el cuadro de información, ya que si el número no estaba centrado o si estaba muy inclinado la red no era capaz de predecir correctamente el número, sin embargo, los resultados en general fueron muy buenos y se obtuvo un resultado correcto en la gran mayoría de los casos. A continuación se presentan una serie de pruebas realizadas para los números desde el 0 hasta el 9.

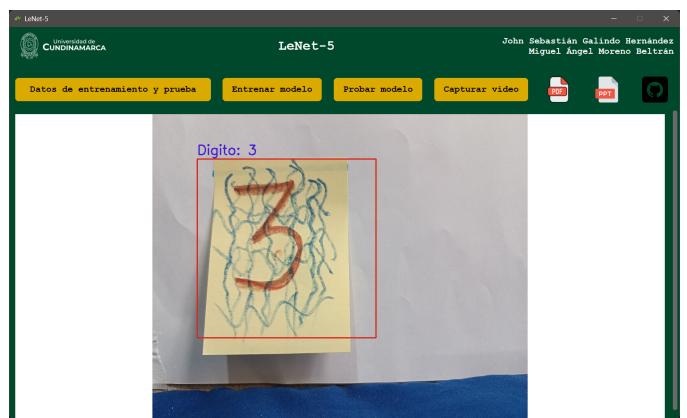


Figura 35. Predicción en tiempo real de la Red LeNet 5 para el número 3

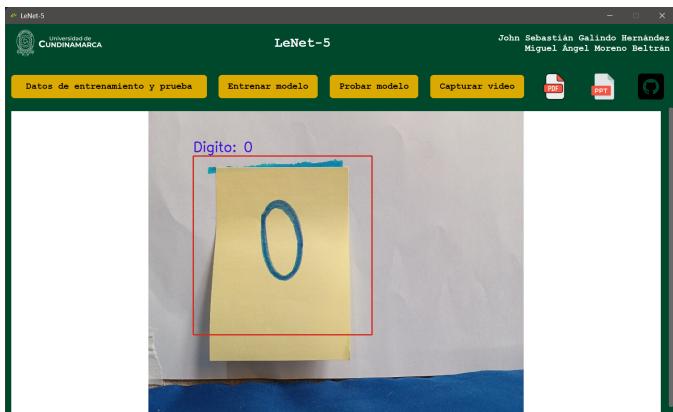


Figura 33. Predicción en tiempo real de la Red LeNet 5 para el número 0

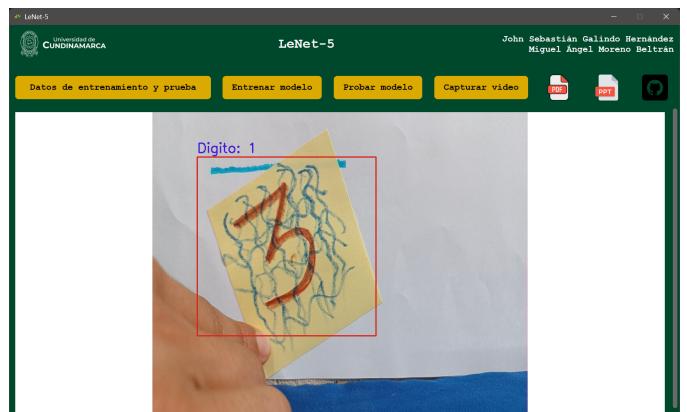


Figura 36. Predicción en tiempo real de la Red LeNet 5 para el número 3 inclinado

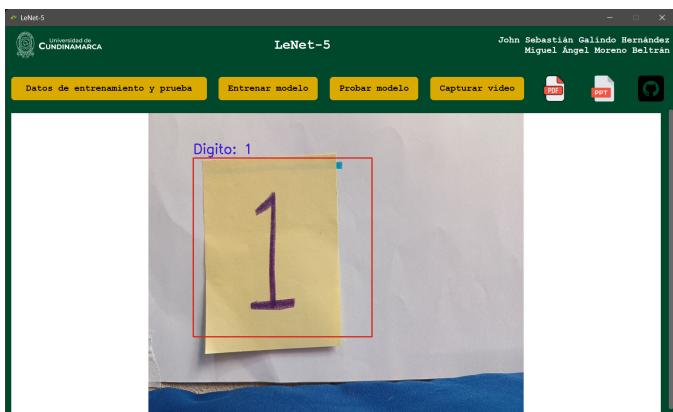


Figura 34. Predicción en tiempo real de la Red LeNet 5 para el número 1

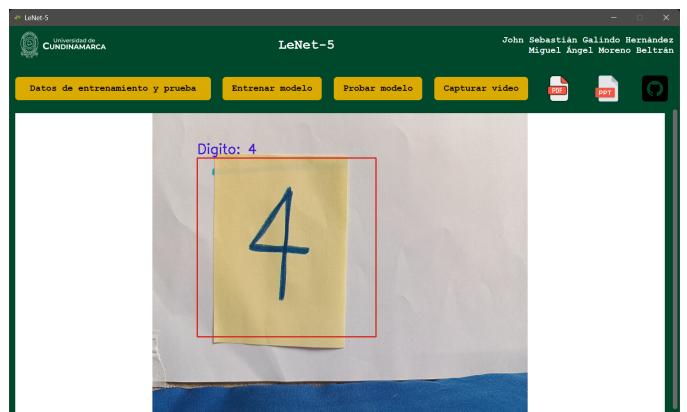


Figura 37. Predicción en tiempo real de la Red LeNet 5 para el número 4

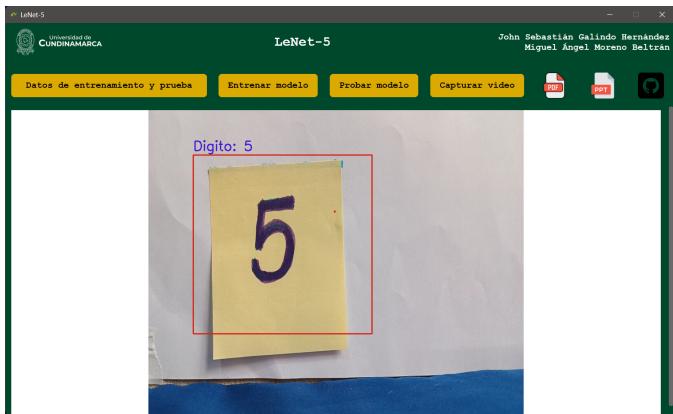


Figura 38. Predicción en tiempo real de la Red LeNet 5 para el número 5

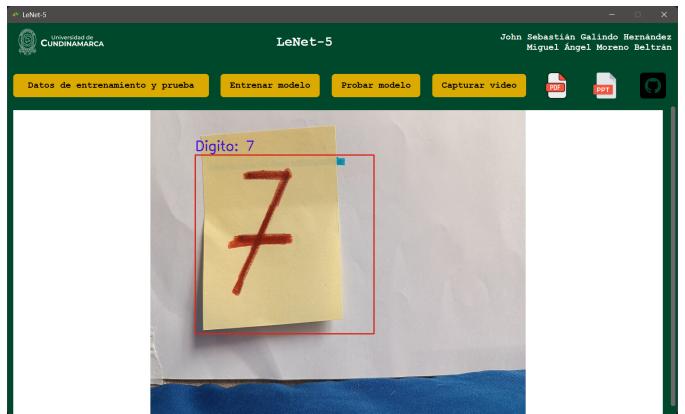


Figura 41. Predicción en tiempo real de la Red LeNet 5 para el número 7

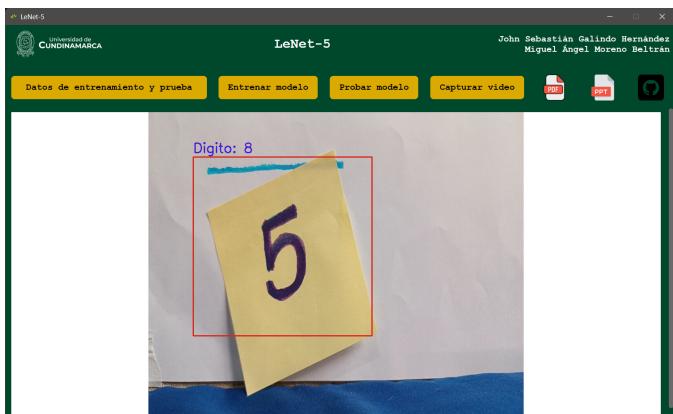


Figura 39. Predicción en tiempo real de la Red LeNet 5 para el número 5 inclinado

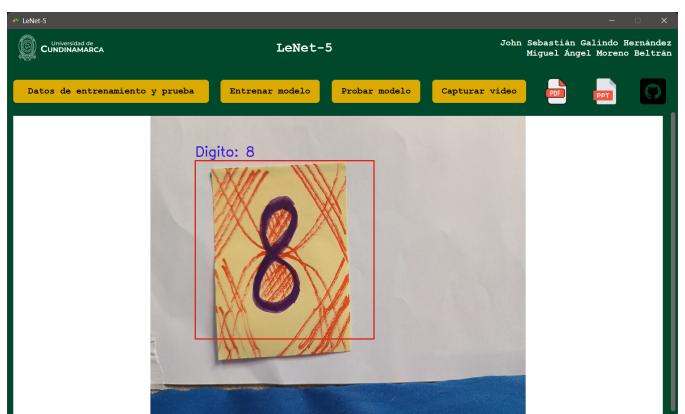


Figura 42. Predicción en tiempo real de la Red LeNet 5 para el número 8

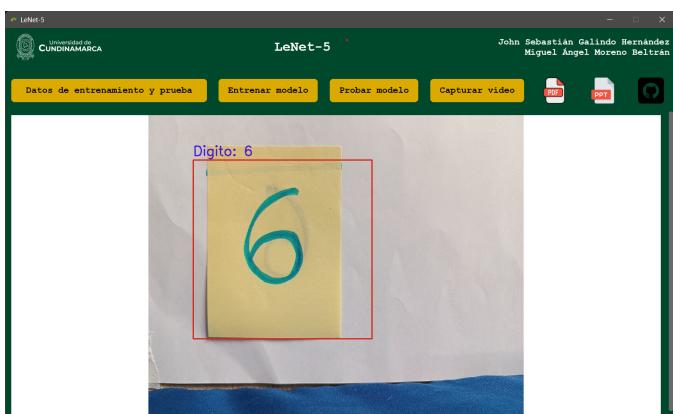


Figura 40. Predicción en tiempo real de la Red LeNet 5 para el número 6

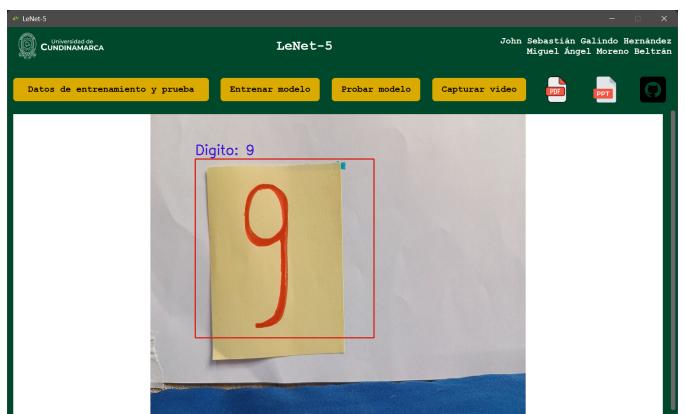


Figura 43. Predicción en tiempo real de la Red LeNet 5 para el número 9

Como se puede ver en las imágenes, se implementó una región de interés (ROI) para que la red solo procese la parte de la imagen que contiene el número, esto se hizo para mejorar la predicción de la red y para que la red no se vea afectada por el ruido de la imagen. Por otro lado, se corrobora que ni el ruido ni el color del número afectan la predicción de la red, ya que se realizaron pruebas con números de diferentes colores y con números con rayones y la red fue capaz de predecir correctamente el número en la gran mayoría de los casos; sin embargo, la red no es perfecta y en algunos casos

no es capaz de predecir correctamente el número, sobre todo cuando el número no esta centrado o cuando el número esta muy inclinado hacia alguno de sus lados, lo que demuestra que los filtros de la red aprendieron a identificar patrones en base a la posición del número en la imagen, lo cual cobra sentido al saber que debe diferenciar entre formas similares pero en diferente posicion como las del 6 y el 9.

V. CONCLUSIONES

A. John Sebastián Galindo Hernández

La red neuronal LeNet es un paso gigante de las redes neuronales artificiales a las redes neuronales convolucionales, ya que estas últimas son capaces de reconocer patrones en imágenes, lo cual es una tarea muy compleja y más aún cuando se tienen muchas categorías a clasificar. La red LeNet es un ejemplo de una CNN implementada con el objetivo de mantener una generalización en el entrenamiento mientras se intenta reducir la información de las imágenes originales solo a los patrones más importantes, esto se logra mediante las capas de su arquitectura. Adentrándose en la opinion sobre como predice la red LeNet, se puede decir que es muy buena, ya que en la mayoría de los casos predice correctamente la categoría de la imagen, sin embargo, en algunos casos que parecen sencillos para un humano, la red falla, esto se puede deber a que los datos de entrenamiento no son suficientes o algunas imagenes tienen un ruido que termina confundiendo a la red. En general, la red LeNet es una herramienta muy útil para clasificar imágenes, pero se debe tener en cuenta que al trabajar con imágenes en tiempo real se debe tener presente un estandar en posicionamiento, tamaño y color de las numeros en las imágenes para que la red pueda clasificar correctamente. Por ultimo, LeNet es una red que obtiene muy bien las características y patrones de los datos de entrada, lo cual la hace maleable para diferentes casos como el de las señales de trafico, las imagenes de cancer de mama, las imágenes de letras, números y simbolos del codigo ASCII o las imágenes de multiples dígitos escritos a mano, en esta ultima, también se puede comprender que la red puede ser juntada con otras técnicas que segmenten la información en una imagen para separar los dígitos y clasificarlos de manera independiente dando así una clasificación más precisa y que se pueda volver a combinar para obtener el número completo.

B. Miguel Ángel Moreno Beltrán

La red neuronal convolucional LeNet destaca por su relevancia en entornos actuales, gracias a su capacidad para generalizar patrones visuales. Aunque es limitada para tareas más complejas, su precisión puede mejorarse mediante técnicas avanzadas, como se observó con el desarrollo de LeNet-5 y su versión mejorada Boosted LeNet-4. En este laboratorio, el uso de funciones de activación modernas como ReLU y el optimizador Adam permitieron alcanzar un rendimiento superior del 93.1 %, en comparación con las configuraciones originales. Para finalizar, LeNet demuestra un rendimiento altamente competitivo en entornos específicos y controlados, como el reconocimiento de números en imágenes, manteniéndose como una arquitectura eficaz y adaptable a diversas aplicaciones.

C. Conclusion Final

LeNet ha demostrado ser una red neuronal convolucional pionera y relevante en el ámbito actual, gracias a su capacidad para reconocer patrones visuales y clasificar imágenes de forma precisa. La arquitectura de LeNet, especialmente en su versión LeNet-5, fue un avance significativo hacia el desarrollo de redes neuronales más complejas y adaptadas a tareas de procesamiento de imágenes. Las mejoras, como el Boosted LeNet-4, y el uso de técnicas modernas como las funciones de activación ReLU y el optimizador Adam, han mostrado que su rendimiento puede alcanzar niveles competitivos, con una precisión superior al 93.1 % en condiciones controladas. Aunque limitada para algunas tareas complejas y entornos no controlados, LeNet sigue siendo una herramienta útil y adaptable a diversas aplicaciones, desde el reconocimiento de números y letras hasta el procesamiento de señales de tráfico y la detección de anomalías en imágenes médicas. Su capacidad para extraer y generalizar patrones la convierte en una opción ideal para proyectos específicos, donde con el apoyo de técnicas complementarias, como la segmentación de imágenes, podría mejorar aún más su precisión y aplicabilidad en la clasificación de imágenes complejas y ruidosas.

REFERENCIAS

- [1] quanta magazine, “Funcionamiento de una red deep learning básica,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://iddigitalschool.com/bootcamps/wp-content/uploads/2021/11/deep-learning-proceso.png>
- [2] IBM. (2024) ¿qué es el deep learning? Consultado: 3 de noviembre de 2024. [Online]. Available: <https://www.ibm.com/es-es/topics/deep-learning>
- [3] J. Cardete, “Imagen en escala de grises,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://images.javatpoint.com/tutorial/dip/images/concept-of-pixel2.png>
- [4] javatpoint, “Representación de una imagen en pixeles,” 2024, consultado: 4 de noviembre de 2024. [Online]. Available: https://miro.medium.com/v2/resize:fit:828/format:webp/0*IJuQtgtes-fU7sLy.png
- [5] J. Cardete. (2024) Convolutional neural networks: A comprehensive guide. Consultado: 3 de noviembre de 2024. [Online]. Available: <https://medium.com/thedeepphub/convolutional-neural-networks-a-comprehensive-guide-5cc0b5ae175>
- [6] M. Saeed. (2021) A gentle introduction to sigmoid function. Consultado: 7 de noviembre de 2024. [Online]. Available: <https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/>
- [7] M. All, “Sigmoid function graph,” consultado: 7 de noviembre de 2024. [Online]. Available: <https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks>
- [8] ——. (2024) Introduction to activation functions in neural networks. Consultado: 7 de noviembre de 2024. [Online]. Available: <https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks>
- [9] ——, “Tanh function graph,” consultado: 7 de noviembre de 2024. [Online]. Available: <https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks>
- [10] ——, “Relu function graph,” consultado: 7 de noviembre de 2024. [Online]. Available: <https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks>
- [11] ——, “Softmax function graph,” consultado: 7 de noviembre de 2024. [Online]. Available: <https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks>
- [12] H. Koyuncu, “Kernel de sobel horizontal y vertical,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://www.researchgate.net/profile/Hakan-Koyuncu-3/publication/352877801/figure/fig2/AS:104068467289293@1625129969712/Sobel-horizontal-and-vertical-kernels.ppm>
- [13] IBM, “Calculo en el proceso de convolución,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://www.ibm.com/content/dam/connectedassets-adobe-cms/worldwide-content/creative-assets/s-migr/ul/g/ed/92/iclh-diagram-convolutional-neural-networks.png>
- [14] A. Jain. (2024) Pooling and their types in cnn. Consultado: 4 de noviembre de 2024. [Online]. Available: <https://medium.com/thedeepphub/pooling-in-convolutional-neural-networks-644e8b2b7a4e>
- [15] ——, “Max pooling en una cnn,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://medium.com/@abhishhekjainindore24/pooling-and-their-types-in-cnn-4a4b8a7a4611>
- [16] ——, “Average pooling en una cnn,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://medium.com/@abhishhekjainindore24/pooling-and-their-types-in-cnn-4a4b8a7a4611>
- [17] R. Pramoditha, “Flatten layer en una cnn,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://towardsdatascience.com/convolutional-neural-network-cnn-architecture-explained-in-plain-english-using-simple-diagrams-e5de17eacc8f>
- [18] I. T. Warrior, “Fully connected layer en una cnn,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://indiantechwarrior.com/fully-connected-layers-in-convolutional-neural-networks/>
- [19] N. Patni. (2024) Historia de lenet. Consultado: 4 de noviembre de 2024. [Online]. Available: <https://www.geeksforgeeks.org/what-is-lenet/>
- [20] N. Vishwakarma. (2024) Lenet history and applications. Consultado: 5 de noviembre de 2024. [Online]. Available: <https://www.analyticsvidhya.com/blog/2023/11/lenet-architectural-insights-and-practical-implementation/>
- [21] Y. LeCun, “Lenet-1,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://sh-tsang.medium.com/paper-brief-review-of-lenet-1-lenet-4-lenet-5-boosted-lenet-4-image-classification-1f5f809dbf17>
- [22] ——, “Lenet-4,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://sh-tsang.medium.com/paper-brief-review-of-lenet-1-lenet-4-lenet-5-boosted-lenet-4-image-classification-1f5f809dbf17>
- [23] Z. Aston, “Lenet-5,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://en.wikipedia.org/wiki/LeNet>
- [24] Y. LeCun, “Boosted lenet-4,” consultado: 4 de noviembre de 2024. [Online]. Available: <https://sh-tsang.medium.com/paper-brief-review-of-lenet-1-lenet-4-lenet-5-boosted-lenet-4-image-classification-1f5f809dbf17>
- [25] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, 1998, la imagen de comparación se encuentra en la página 12.
- [26] S. Balasubramaniam, Y. Velmurugan, D. Jaganathan, and S. Dhanasekaran, “A modified lenet cnn for breast cancer diagnosis in ultrasound images,” *MDPI*, vol. 13, no. 2746, 2023.
- [27] A. Gozhulovskyi. (2022) Classification of traffic signs with lenet-5 cnn. Consultado: 5 de noviembre de 2024. [Online]. Available: <https://towardsdatascience.com/classification-of-traffic-signs-with-lenet-5-cnn-cb861289bd62>
- [28] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning nips workshop on deep learning and unsupervised feature learning 2011,” consultado: 6 de noviembre de 2024. [Online]. Available: <http://ufldl.stanford.edu/housenumbers>
- [29] Dive Into Deep Learning, “Convolutional neural network lenet,” consultado: 7 de noviembre de 2024. [Online]. Available: https://d2l.ai/chapter_convolutional-neural-networks/lenet.html
- [30] T. Oliveira. (2021) Numerical images. Consultado: 7 de noviembre de 2024. [Online]. Available: <https://www.kaggle.com/datasets/pintowar/numerical-images>