Inteligencia Computacional

Redes Neuronales

# Reconocimiento óptico de caracteres

## MNIST

**Autor**
Alexander Collado Rojas


Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Diciembre de 2024

# Índice general

# Capítulo 1

# Introducción

## 1.1. Red Neuronal Artificial

Las redes neuronales son sistemas compuestos por elementos interconectados, inspirados en los sistemas nerviosos biológicos. Su principal función es generar un patrón de salida a partir de un patrón de entrada. Un ejemplo común es la clasificación de patrones, como reconocer letras o números escritos a mano, incluso cuando estos varían en forma o estilo. Estas redes son capaces de asignar los patrones de entrada a clases específicas basándose en su similitud con ejemplos aprendidos previamente, resolviendo problemas que involucran reconocimiento de patrones y toma de decisiones. [1]

### 1.1.1. Red Neuronal Convolutiva

Las redes neuronales convolutivas (CNNs) son un subtipo de redes neuronales artificiales que, al igual que estas, están compuestas por neuronas que se optimizan mediante aprendizaje. Cada neurona recibe una entrada y realiza operaciones como el producto escalar seguido de una función no lineal, procesando datos desde vectores de imágenes crudas hasta puntajes finales de clasificación. [2]

Las redes neuronales convolutivas están compuestas por una arquitectura específica que combina capas convolucionales, de pooling y totalmente conectadas, lo que les permite procesar datos de tipo imagen de manera eficiente. Estas redes son particularmente efectivas para el reconocimiento de patrones en imágenes porque aprovechan características como la estructura espacial

de los datos.

En las CNNs, las capas convolucionales utilizan filtros que se aplican sobre pequeñas regiones de la imagen para extraer características locales, como bordes o texturas. Estas características se combinan progresivamente a medida que los datos avanzan a través de la red, generando representaciones más abstractas y útiles para la clasificación. Además, las capas de pooling reducen la dimensionalidad de las representaciones, disminuyendo los parámetros y la complejidad computacional, mientras que las capas totalmente conectadas producen las salidas finales, como las probabilidades de clasificación.

Una de las principales ventajas de las redes neuronales convolutivas es su capacidad para reducir el riesgo de overfitting en comparación con las redes neuronales normales.

- **Conectividad local:** En lugar de conectar cada neurona a todas las neuronas de la capa anterior, como ocurre en las redes tradicionales, las CNNs limitan las conexiones a pequeñas regiones locales de la entrada. Esto reduce significativamente el número de parámetros del modelo, lo que disminuye la complejidad y, por ende, el riesgo de sobreajuste.

- **Compartición de parámetros:** Las CNNs utilizan filtros compartidos que aplican los mismos pesos a diferentes regiones de la imagen, lo que no solo disminuye el número total de parámetros, sino que también mejora la capacidad del modelo para generalizar, ya que aprende características comunes que son relevantes en distintas partes de la entrada.

Estas propiedades, combinadas con técnicas adicionales como el uso de capas de pooling y regularización (como Dropout), permiten que las CNNs mantengan un equilibrio adecuado entre la capacidad del modelo y la generalización, mitigando efectivamente los efectos del overfitting

### 1.1.2. Overfitting

El overfitting es un problema común en el aprendizaje supervisado, que ocurre cuando un modelo se ajusta demasiado bien a los datos de entrenamiento, pero tiene un rendimiento pobre en los datos de prueba. Esto sucede porque el modelo no solo aprende los patrones generales de los datos, sino también el ruido y las peculiaridades específicas del conjunto de entrenamiento, lo que dificulta su capacidad de generalización. En otras palabras, el modelo memoriza los datos de entrenamiento en lugar de captar las relaciones subyacentes entre las variables [3]

## 1.2. MNIST

El problema de MNIST consiste en clasificar dígitos escritos a mano utilizando redes neuronales. Este conjunto de datos estándar contiene 60,000 imágenes para entrenamiento y 10,000 para prueba, todas normalizadas a un tamaño fijo de 28x28 píxeles. Las imágenes están preprocesadas, centradas y segmentadas para facilitar la comparación de resultados entre diferentes técnicas y reducir la carga de procesamiento inicial.

El objetivo es asignar cada imagen de entrada, representada como un vector de 784 características, a una de las 10 clases correspondientes a los dígitos del 0 al 9.

Construir una máquina capaz de leer números escritos a mano es un desafío debido a la gran variabilidad en cómo puede representarse cada número. Por ejemplo, un "3"puede escribirse de muchas formas diferentes en tamaño, estilo o inclinación. Para resolver esto, se agrupan todas las variantes posibles de un número en una misma clase. El sistema, en lugar de buscar una coincidencia exacta, determina a qué clase pertenece el número basado en su similitud con ejemplos previos. Este enfoque permite que un número escrito a mano pueda ser identificado correctamente y procesado como un número impreso. Las redes neuronales son ideales para esta tarea, ya que actúan como clasificadores de patrones, resolviendo problemas complejos como reconocer números independientemente de sus variaciones. [4]

# Capítulo 2

# Solución

## 2.1. Estructura del Proyecto

El proyecto está organizado en tres directorios principales:

- **data:** Este directorio contiene las imágenes y las etiquetas necesarias tanto para el entrenamiento como para las pruebas del modelo.
- **src:** quí se encuentra el código fuente relacionado con el procesamiento y la implementación del modelo:
  - **read.py:** Código encargado de leer los archivos de imágenes y etiquetas ubicados en el directorio data.
  - **main_convolutiva.py:** Contiene el código de implementación de la red neuronal convolutiva, desarrollada utilizando las bibliotecas TensorFlow y Keras. Este archivo se encargará de entrenar el modelo, evaluar su rendimiento y guardar el modelo resultante. Se describirá más en detalle en secciones posteriores.
  - **main_salida.py:** Código que permite cargar un modelo previamente entrenado y realizar predicciones sobre el conjunto de pruebas y de entrenamiento. Además, este script calcula las tasas de error y genera las etiquetas predichas por el modelo entrenado.
- **modelos:** En este directorio se almacena el mejor modelo entrenado, guardado en formato .h5. También contiene el archivo cadena_salida.txt,

que guarda las 10,000 etiquetas predichas por el modelo para el conjunto de pruebas, representando los dígitos que la red neuronal identificó.

```
Proyecto/
|-- data/
| |-- t10k-images.idx3-ubyte
| |-- t10k-labels.idx1-ubyte
| |-- train-images.idx3-ubyte
| |-- train-labels.idx3-ubyte
|-- src/
| |-- read.py
| |-- main_convolutiva.py
| |-- main_salida.py
|-- modelo/
| |-- model_convolutiva_43.h5
| |-- cadena_salida.txt
```

## 2.2. Soluciones Exploradas

Durante el desarrollo de la práctica, he explorado diferentes soluciones como indica el guión de las prácticas, estas soluciones también se implementaron usando **tensorflow** y **keras**:

- **Red Neuronal Simple:** Es la aproximación más básica explorada en esta práctica. Consiste en una única capa que recibe las imágenes de entrada con dimensiones de 28x28 píxeles. Estas imágenes son procesadas mediante una capa de salida con 10 neuronas. La activación utilizada en esta capa es **softmax**, que permite calcular las probabilidades de cada clase para realizar la clasificación.

  El modelo utiliza el inicializador **glorot_uniform** para los pesos, que distribuye los valores iniciales de manera uniforme dentro de un rango calculado en función del número de neuronas de entrada y salida. Esto mejora la estabilidad del entrenamiento al evitar gradientes demasiado grandes o pequeños.

  El entrenamiento se realizó utilizando el optimizador **Adam** con un **learning rate** de 0.001, configurando **20 épocas** con un **batch_size de 32**. Para evitar el sobreajuste, se empleó la técnica de EarlyStopping, configurada con una paciencia de **3 épocas** consecutivas sin mejora en la pérdida de validación.

### 2.2.1. ¿Qué es Accuracy y Loss?

El accuracy mide el porcentaje de predicciones correctas realizadas por el modelo sobre el total de datos evaluados (10 000). Específicamente, indica cuán bien el modelo clasifica las entradas en sus categorías correspondientes.

La pérdida evalúa que tan lejos están las predicciones del modelo de los valores reales durante el entrenamiento.

El accuracy aumenta rápidamente en las primeras cuatro épocas, superando el 92 %, y luego se estabiliza en torno al 92.6 %. Esto refleja que el modelo ha alcanzado su máximo rendimiento con la arquitectura actual.

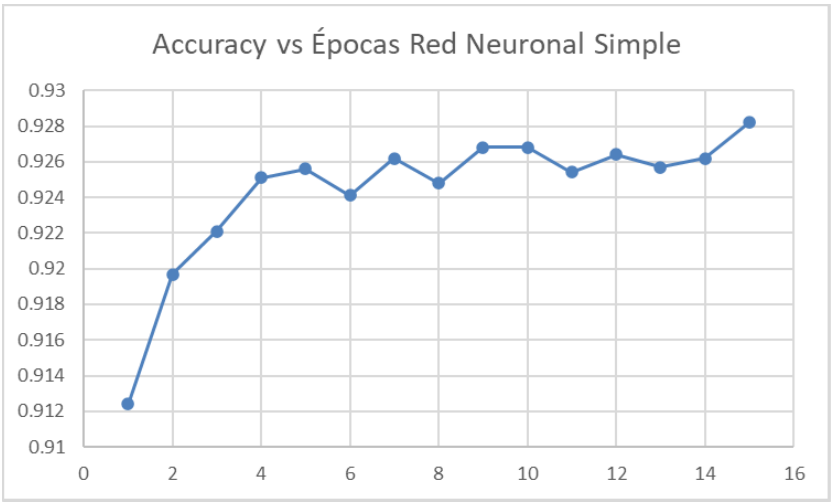


Figura 2.1: Gráfica Accuracy vs Épocas

La pérdida disminuye rápidamente en las primeras épocas, indicando un aprendizaje eficiente del modelo. A partir de la quinta época, la pérdida se estabiliza, lo que sugiere que el modelo ha convergido y no mejora significativamente con más entrenamiento.

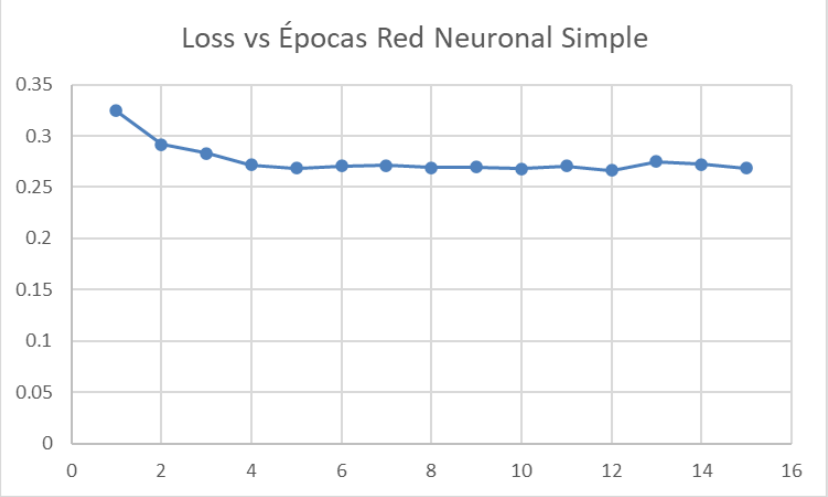


Figura 2.2: Gráfica Loss vs Épocas

- **Red Neuronal Multicapa:** Esta red neuronal amplía la complejidad del modelo simple al añadir una capa oculta, lo que permite capturar relaciones más complejas en los datos. Las imágenes de entrada se aplastan a un vector unidimensional mediante una capa de entrada. Posteriormente, pasan a una capa oculta con **256 neuronas**, utilizando la función de activación **sigmoid**, que introduce no linealidad en el modelo. Los pesos de esta capa se inicializan con el método **lecun_uniform**, optimizado para este tipo de redes.

  La capa de salida contiene **10 neuronas** con activación **softmax**, como en el modelo anterior, para clasificar las imágenes en una de las 10 clases posibles. El entrenamiento se realizó con el optimizador **Adam** y un **learning rate** de 0.001, configurando **20 épocas** con un **batch_size de 32**. También se implementó EarlyStopping, con una paciencia de **5 épocas** consecutivas sin mejora en la pérdida de validación.

La precisión del modelo aumenta rápidamente durante las primeras cinco épocas, alcanzando valores cercanos al 98 %. Esto demuestra que el modelo aprende de manera eficiente en las primeras etapas del entrenamiento. A partir de la quinta época, la precisión se estabiliza con ligeras oscilaciones, manteniéndose en torno al 98.2 %

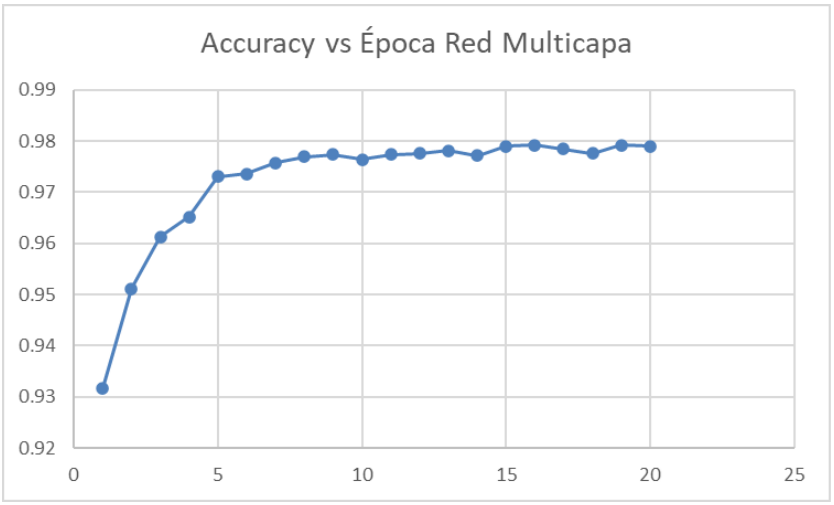


Figura 2.3: Gráfica Loss vs Épocas

La pérdida disminuye drásticamente durante las primeras épocas, especialmente entre la primera y la quinta, lo que indica que el modelo ajusta sus pesos de manera eficiente para reducir el error. Después de la quinta época, la pérdida se estabiliza cerca de un valor bajo de 0.05, con pequeñas fluctuaciones en las épocas posteriores.

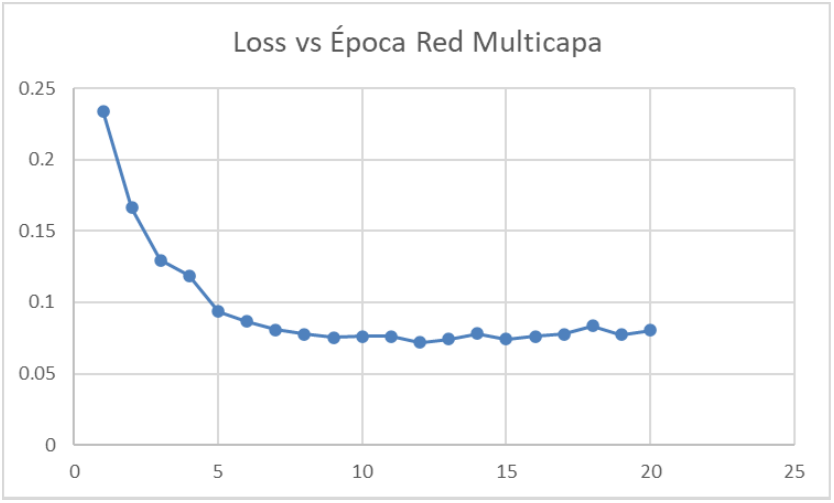


Figura 2.4: Gráfica Loss vs Épocas

La red multicapa supera significativamente a la red simple en términos de rendimiento. Mientras que la precisión final de la red simple alcanza alre-

dedor del 92.6 %, la red multicapa logra una precisión notablemente mayor, alrededor del 98.2 %. Esto se debe a la inclusión de una capa oculta, que permite al modelo aprender patrones más complejos en los datos.

## 2.3. Mejor Solución

La mejor solución que he encontrado implementa una **Deep Network** diseñada con una topología compleja de 4 capas ocultas utilizando Python en conjunto con las librerías Tensorflow y Keras.

### 2.3.1. Conceptos

A continuación se describirán algunos conceptos necesarios para entender sobre la topología y el entrenamiento de la red neuronal.

- **Filtros:** Es una pequeña matriz utilizada en las Redes Neuronales Convolutivas para extraer características de las imágenes, desplazándose sobre ellas y combinándose con las partes subyacentes para obtener el valor máximo de cada convolución [5]

- **Tamaño del Kernel:** Determina el campo receptivo de cada operación de convolución. Los kernels más pequeños capturan información más local, como bordes o pequeños patrones, mientras que los kernels más grandes capturan información más global, como formas completas o estructuras más amplias en la imagen [6]

- **Función de Activación:** Funciones matemáticas aplicadas a cada neurona de la red neuronal para determinar si debe .ªctivarse"(dispararse) en función de su entrada. Sin estas funciones, las redes neuronales solo realizarían transformaciones lineales, lo que limitaría su capacidad para aprender patrones y relaciones complejas en los datos. Al introducir no linealidad, las funciones de activación permiten que el modelo capture y represente mejor las características intrínsecas de los datos [7]

- **Inicializador de Pesos:** Determinan los valores iniciales de los pesos en una red neuronal antes de comenzar el proceso de entrenamiento. La elección de un inicializador adecuado es crucial, ya que influye en la convergencia y el rendimiento del modelo. Inicializaciones inapropiadas pueden llevar a problemas como la divergencia o ralentización del entrenamiento debido a la explosión o desaparición de gradientes [8]

- **Épocas:** Hiperparámetro que define cuántas veces el algoritmo de aprendizaje recorrerá todo el conjunto de datos de entrenamiento. Una época implica que cada muestra del conjunto de entrenamiento ha tenido la oportunidad de contribuir a la actualización de los parámetros internos del modelo [9]

- **Batch Size:** Hiperparámetro que define la cantidad de muestras de entrenamiento que el algoritmo procesa antes de actualizar los parámetros internos del modelo. En el contexto del descenso de gradiente, el tamaño del lote influye en la precisión de la estimación del gradiente del error [9]:

  - **Lotes pequeños:** Proporcionan estimaciones más ruidosas del gradiente, lo que puede conducir a una convergencia más rápida pero menos estable.

  - **Lotes grandes:** Ofrecen estimaciones más precisas del gradiente, resultando en una convergencia más estable pero potencialmente más lenta.

- **Learning Rate:** Hiperparámetro que controla cuánto se ajustan los pesos del modelo en respuesta al error estimado cada vez que se actualizan. En términos matemáticos, al usar gradiente descendente estocástico, el learning rate se multiplica por el gradiente de la función de pérdida para determinar el tamaño de los pasos hacia un mínimo de la función de pérdida durante la optimización [10]

- **Momentum:** Técnica utilizada en la optimización de redes neuronales que acelera el proceso de convergencia y ayuda a evitar quedarse atrapado en mínimos locales. Funciona acumulando una fracción de la actualización previa de los pesos y sumándola a la actualización actual [11]

- **Regularización:** Técnica que se utiliza para prevenir el sobreajuste en los modelos. Se logra penalizando los parámetros del modelo o introduciendo aleatoriedad

- **Optimizador:** Componente esencial en las redes neuronales, encargados de actualizar los pesos del modelo durante el proceso de entrenamiento. Son algoritmos diseñados para encontrar el conjunto óptimo de pesos que minimice la función de pérdida, guiando al modelo hacia la solución más precisa y eficiente

- **Pool Size:** Parámetro que define las dimensiones de la ventana utilizada en la operación de pooling. Especifica el área sobre la cual se aplicará la reducción de dimensionalidad

### 2.3.2. Topología

#### Capa de Entrada

Según el guion de prácticas, las imágenes de entrada son *imágenes normalizadas de 28x28 píxeles*. La dimensión de entrada se configura como (28, 28, 1) indicando las dimensiones espaciales de las imágenes y que estas están en escala de grises, es decir con un canal de color 1.

#### Primera Capa Convolutiva

La primera capa oculta de la red es una capa convolutiva. En esta capa se configuran los siguientes parámetros:

- **Filtros:** Se han utilizado **64 filtros** para permitir que el modelo aprenda una amplia variedad de patrones presentes en las imágenes, como bordes, texturas y formas básicas. Este número equilibra la capacidad del modelo para capturar características significativas sin introducir una complejidad computacional excesiva.

- **Función de Activación (ReLU)**: La función de activación **Unidad Lineal Rectificada** es utilizada porque introduce no linealidad en el modelo, permitiendo que aprenda relaciones complejas en los datos. Además, es computacionalmente eficiente y ayuda a evitar el problema del desvanecimiento del gradiente.

- **Inicializador del Kernel:** El inicializador **he_normal** se utiliza para asignar valores iniciales a los pesos de la capa. Este inicializador está optimizado para activaciones ReLU, distribuyendo los pesos de manera que se evite la explosión o desaparición de gradientes, asegurando estabilidad en el entrenamiento.

#### Batch Normalization

En esta red neuronal, la normalización por lotes mejora la eficiencia y estabilidad del entrenamiento al normalizar las activaciones de cada capa. Esto asegura que las siguientes capas analicen datos con distribuciones consistentes, facilitando la convergencia al usar el optimizador SGD. Además,

reduce las oscilaciones del gradiente y permite que el modelo ajuste los pesos de manera más efectiva, acelerando el proceso de optimización y mejorando el rendimiento general del modelo [12].

### MaxPooling2D

Se utiliza para reducir las dimensiones espaciales de las activaciones en la red, haciendo la representación más compacta y eficiente. Esto permite disminuir la cantidad de parámetros, mantener las características más relevantes, y mejorar la invariancia frente a cambios de escala y orientación en las imágenes, además de ayudar a prevenir el sobreajuste.

### Dropout

El Dropout con una tasa de 0.3 se utiliza después de la primera capa convolutiva como una técnica de regularización para prevenir el sobreajuste. Esta capa desactiva aleatoriamente el 30 % de las conexiones durante el entrenamiento, lo que obliga al modelo a no depender excesivamente de características específicas aprendidas por la primera capa.

### Segunda Capa Convolutiva

Se utilizan **128 filtros** en la segunda capa convolutiva para permitir que el modelo aprenda características más complejas y abstractas basadas en las características básicas extraídas en la primera capa. Mientras que la primera capa detecta patrones simples como bordes o texturas, esta segunda capa se encarga de combinar esos patrones básicos para identificar estructuras más avanzadas en las imágenes.

La capa sigue utilizando la función de activación **ReLU**, que introduce no linealidad y mejora la eficiencia del entrenamiento al evitar el desvanecimiento del gradiente. También se mantiene el inicializador de kernel **he_normal**, optimizado para activaciones ReLU, lo que garantiza estabilidad en la propagación de los gradientes y una convergencia más eficiente.

### BatchNormalization, MaxPooling2D y Dropout

Se utiliza nuevamente **BatchNormalization** y **MaxPooling2D**, tal como se explicó anteriormente.

No se aplica **Dropout** en esta capa, ya que el modelo ya cuenta con regularización en la primera capa a través de un Dropout de 0.3. En esta

etapa, el modelo se enfoca en consolidar las características más abstractas aprendidas, permitiendo que la regularización previa y la reducción de dimensionalidad mediante MaxPooling2D controlen el riesgo de sobreajuste de manera adecuada sin comprometer la capacidad de aprendizaje del modelo.

### Flatten

La capa Flatten se refiere al proceso de convertir todas las características multidimensionales en un vector unidimensional. Esto se realiza generalmente antes de pasar la salida de las capas convolutivas a las capas completamente conectadas de la red [13].

### Primera Capa Completamente Conectada

Se han utilizado **128 neuronas** en esta capa para permitir que la red combine las características extraídas por las capas convolutivas y genere una representación más abstracta y significativa de los datos. Usar demasiadas neuronas podría incrementar innecesariamente la complejidad del modelo, mientras que muy pocas podrían limitar su capacidad para capturar patrones relevantes en las características extraídas.

A diferencia de otras funciones de activación como sigmoid o tanh, ReLU ayuda a prevenir el problema del desvanecimiento del gradiente, lo que facilita el aprendizaje en redes profundas.

El inicializador he_normal se utiliza para asignar valores iniciales a los pesos de las capas, optimizados específicamente para activaciones como ReLU.

### BatchNormalization

El uso de BatchNormalization después de esta capa asegura que las características combinadas se procesen de manera eficiente y consistente, mejorando la capacidad del modelo para generalizar en datos nuevos.

### Segunda Capa Completamente Conectada

Se han utilizado **256 neuronas** en esta capa para ampliar la capacidad de representación del modelo. Al ser la última capa oculta antes de la capa de salida, esta capa tiene como objetivo refinar las características combinadas aprendidas previamente y prepararlas para la clasificación final.

**BatchNormalization:**

Nuevamente se usa BatchNormalization para tener una distribución estable de las activaciones.

**Dropout**

El Dropout del 20 % justo antes de la capa de salida se utiliza para mejorar la capacidad del modelo de generalizar en datos nuevos y prevenir el sobreajuste.

**Capa de Salida**

El número de **10 neuronas** en la capa de salida corresponde al número de clases posibles en el problema de clasificación, en este caso, las 10 posibles cifras del conjunto de datos MNIST. Cada neurona en la capa de salida representa una clase y su salida es la probabilidad de que la imagen de entrada pertenezca a esa clase.

La función de activación **softmax** se utiliza en la capa de salida para convertir las activaciones de las neuronas en probabilidades. Softmax toma los valores de salida de las neuronas y los normaliza, de manera que todas las salidas suman 1 y representan la probabilidad de cada clase. Esta es la función de activación estándar para problemas de clasificación multiclase, como en MNIST, ya que facilita la asignación de una única clase con la mayor probabilidad.

El inicializador **glorot_uniform** se utiliza para distribuir los pesos de manera uniforme dentro de un rango determinado. Está diseñado para funcionar bien con funciones de activación como softmax. El uso de glorot_uniform garantiza que las activaciones no sean ni demasiado grandes ni demasiado pequeñas, evitando problemas de explosión o desvanecimiento de gradientes

### 2.3.3. Configuración del Modelo

**Optimizador**

El SGD (Descenso de Gradiente Estocástico) es una técnica de optimización utilizada en redes neuronales que simplifica el cálculo del gradiente al estimarlo en cada iteración con base en un único ejemplo tomado aleatoriamente del conjunto de datos de entrenamiento. Esto lo diferencia del Descenso de Gradiente tradicional, que calcula el gradiente promedio sobre todos los ejemplos en cada iteración [14] [15].

- **Eficiencia:** Al calcular el gradiente usando un solo ejemplo por iteración, SGD es más rápido y requiere menos memoria que calcular el gradiente completo en cada paso.
- **Ruido:** Aunque introduce ruido debido a la aleatoriedad en la selección de ejemplos, este ruido puede ayudar al algoritmo a escapar de mínimos locales y a buscar un mínimo global.
- **Dependencia del learning rate:** El tamaño de los pasos dados por el algoritmo hacia el mínimo de la función de pérdida está controlado por el hiperparámetro learning rate, que debe ajustarse cuidadosamente para un aprendizaje eficiente.

Se utiliza un **learning rate** de 0.01 porque ofrece un equilibrio adecuado entre la velocidad de aprendizaje y la estabilidad, permitiendo una convergencia eficiente sin causar oscilaciones ni divergencia. Por otro lado, un **momentum** de 0.95 acumula gradientes previos para mantener actualizaciones consistentes, reduciendo oscilaciones y acelerando el proceso de convergencia hacia el mínimo de la función de pérdida.

#### Función de Pérdida

La función de pérdida se encarga de medir la diferencia entre las predicciones realizadas por el modelo y las etiquetas reales. En este caso, se utiliza **categorical_crossentropy**, que es adecuada para problemas de clasificación como MNIST. Esta función penaliza de manera más severa las predicciones incorrectas que están muy alejadas de la clase correcta, lo que ayuda al modelo a ajustar sus pesos de forma que aprenda probabilidades precisas y mejore su capacidad de clasificación.

#### Métricas

**Accuracy** es una métrica que calcula el porcentaje de predicciones correctas del modelo. Es una métrica directa y comprensible para evaluar el rendimiento de un modelo en problemas de clasificación.

### 2.3.4. Entrenamiento

#### Image Data Generator

El uso de ImageDataGenerator para la aumentación de datos en el conjunto MNIST se justifica por su capacidad para generar, en tiempo real,

variaciones de las imágenes de entrenamiento mediante transformaciones como rotaciones, zooms y desplazamientos. Esto enriquece el conjunto de datos, permitiendo que el modelo aprenda a reconocer patrones bajo diferentes condiciones, lo que mejora su capacidad de generalización y robustez frente a variaciones en datos no vistos previamente. Además, ImageDataGenerator realiza estas transformaciones de manera eficiente durante el proceso de entrenamiento, sin necesidad de almacenar físicamente las imágenes aumentadas, optimizando el uso de recursos computacionales [16].

Sin embargo, es importante tener cuidado con las rotaciones. Si se utilizan valores muy altos, las imágenes pueden distorsionarse al punto de que los números no sean identificables correctamente, lo que puede afectar el rendimiento del modelo. Por ejemplo, un "6"podría interpretarse como un "9", y viceversa, generando ambigüedad en el aprendizaje y afectando negativamente la precisión del modelo.

**Configuración del Proceso de Entrenamiento**

- **Datos de Entrenamiento:** Las imágenes de entrenamiento son procesadas en mini-lotes, definidos por el parámetro batch_size de tamaño 32
- **Épocas:** Se configura el modelo para entrenar durante 50 épocas, permitiendo múltiples iteraciones sobre los datos de entrenamiento. **Validación:** Se reserva un subconjunto del 20 % de los datos de entrenamiento para evaluar el rendimiento del modelo en cada época. Esto permite monitorear el comportamiento del modelo en datos que no está viendo directamente durante el entrenamiento. **Pasos por Época:** Define el número de iteraciones necesarias para procesar todo el conjunto de datos en una época. Se calcula dividiendo el número total de muestras de entrenamiento (48 000) por el tamaño del lote (batch_size) (32).

### 2.3.5. Callbacks

Los callbacks son herramientas que se ejecutan durante el entrenamiento para realizar tareas específicas, como detener el entrenamiento o ajustar la tasa de aprendizaje. En este caso, se utilizan:

**EarlyStopping**

Monitorea la pérdida en el conjunto de validación y detiene el entrenamiento si no hay mejora después de un número determinado de épocas consecutivas, que para mi implementación se definió como paciencia 8. Esto previene el entrenamiento innecesario y el sobreajuste.

**ReduceLROnPlateau**

El callback ReduceLROnPlateau se utiliza en la red convolutiva para ajustar dinámicamente la tasa de aprendizaje durante el entrenamiento, especialmente al usar el optimizador SGD, que depende de un learning rate bien ajustado. Este callback reduce la tasa de aprendizaje cuando la pérdida en el conjunto de validación deja de mejorar después de 2 épocas consecutivas, como se define en el parámetro patience=2. La reducción se realiza multiplicando la tasa de aprendizaje actual por un factor de 0.7, asegurando ajustes graduales y efectivos. Además, se establece un límite mínimo para la tasa de aprendizaje, evitando que el modelo deje de aprender completamente en etapas avanzadas.

### 2.3.6. Evaluación

Por último, el modelo evalúa tanto el conjunto de entrenamiento como el conjunto de prueba, calculando la tasa de error en cada uno. Esto se realiza utilizando las funciones model.evaluate para obtener la pérdida y la precisión en ambos conjuntos. A partir de la precisión, se calcula la tasa de error como el complemento porcentual de la misma.

$$100 - (\text{accuracy} \times 100)$$

Esto proporciona una medida clara del desempeño del modelo tanto en los datos utilizados para el entrenamiento como en los datos reservados para evaluar su capacidad de generalización.

Además, el modelo entrenado se guarda en un archivo, permitiendo su reutilización para realizar predicciones y generar la cadena de resultados. Este proceso se lleva a cabo en el script main_salida.py, donde el modelo previamente guardado se carga para realizar predicciones y evaluar su rendimiento en tareas específicas.

### 2.3.7. Resultados

El mejor resultado obtenido durante los experimentos ha sido una tasa de error sobre el conjunto de prueba del 0.43 % y una tasa de error sobre el conjunto de entrenamiento del 0.13 %. La cadena resultante del conjunto MNIST se ha enviado a través del formulario de Google.

A continuación, se presentan las gráficas de épocas vs accuracy y épocas vs loss sobre el conjunto de prueba, que ilustran cómo ha evolucionado el aprendizaje del modelo durante el entrenamiento. Estas gráficas se han generado a partir de un entrenamiento posterior en el que se alcanzó una tasa de error mínima de 0.47 % sobre el conjunto de prueba.

Esta ligera variación en los resultados, en comparación con el mejor obtenido (0.43 %), se debe principalmente a la aleatoriedad en la selección del conjunto de entrenamiento, así como a otros factores inherentes al proceso de entrenamiento, como la inicialización de los pesos y las transformaciones realizadas durante el aumento de datos.

Para alcanzar una tasa de error del 0.43 % realicé múltiples ejecuciones, ajustando diversos parámetros como la tasa de aprendizaje y los filtros, además de probar técnicas como callbacks y data augmentation. Sin embargo, el resultado documentado en esta memoria, utilizando el código **main_convolutiva.py**, fue el que finalmente permitió obtener este porcentaje, con todos los detalles y configuraciones reflejados en el texto.

### Gráficas

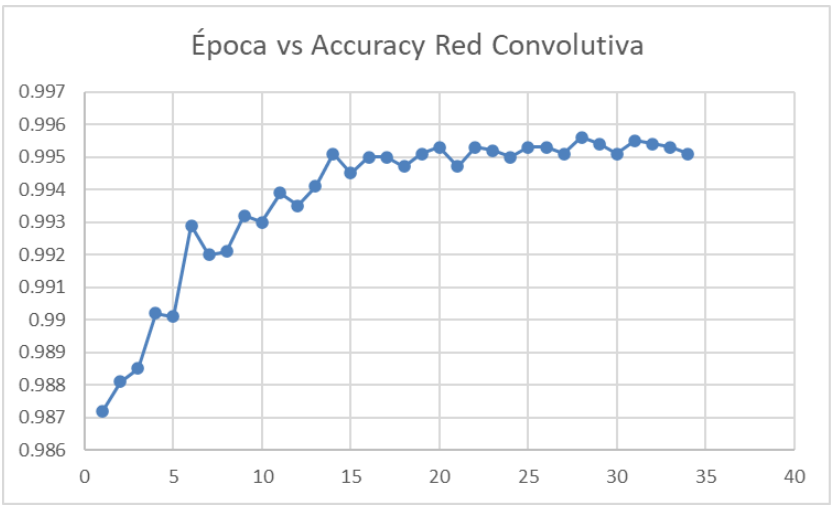


Figura 2.5: Gráfica Accuracy vs Épocas

La precisión aumenta rápidamente en las primeras épocas, alcanzando valores superiores al 99 % antes de la época 10. Esto confirma que el modelo aprende de manera eficiente las características necesarias para clasificar correctamente los datos.

Después de la época 10, la precisión se estabiliza cerca de 99.5 %, lo que indica que el modelo alcanza su límite de rendimiento en esta configuración

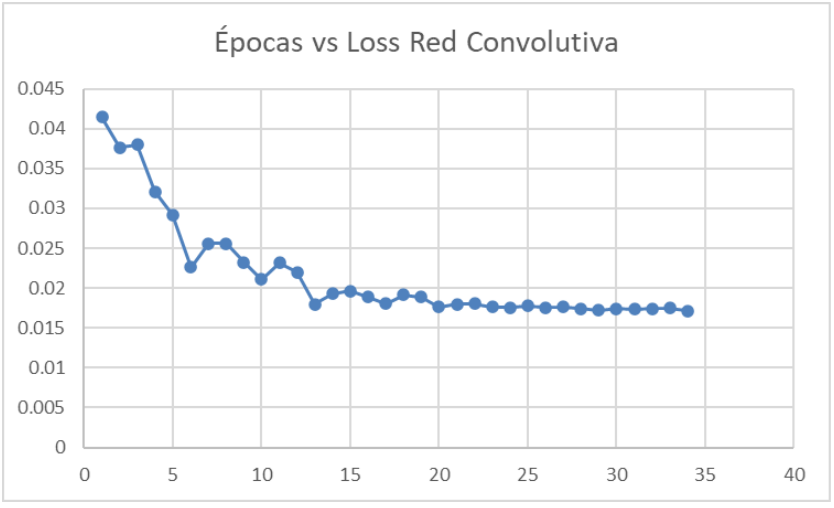


Figura 2.6: Gráfica Loss vs Épocas

La pérdida disminuye significativamente durante las primeras 10 épocas, indicando que el modelo aprende rápidamente los patrones principales de los datos. Posteriormente, la reducción del loss se estabiliza, reflejando que el modelo está alcanzando su capacidad óptima.

A partir de la época 15 aproximadamente, el loss se estabiliza cerca de un valor bajo constante, lo que sugiere que el modelo ha convergido. Indicando que las configuraciones de regularización, optimización y callbacks funcionan bein.

# Bibliografía


[1] P. Picton. What is a neural network? pages 1–12. Macmillan Education UK, 1994.

[2] K. O'Shea. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[3] X. Ying. An overview of overfitting and its solutions. In *Journal of Physics: Conference Series*, volume 1168, page 022022. IOP Publishing, February 2019.

[4] W. Zhu. Classification of mnist handwritten digit database using neural network. In *Proceedings of the Research School of Computer Science*, Acton, ACT, 2601, 2018.

[5] Convolutional filter. URL `https://www.sciencedirect.com/topics/computer-science/convolutional-filter#:~:text=A%20Convolutional%20Filter%20is%20a,maximum%20value%20from%20each%20convolution`. Accessed: 2024-12-08.

[6] How to choose kernel size in cnn, . URL `https://www.geeksforgeeks.org/how-to-choose-kernel-size-in-cnn/`. Accessed: 2024-12-08.

[7] Activation functions in convolutional neural networks. URL `https://medium.com/@vishnuam/activation-functions-in-convolutional-neural-networks-cnn-b1114a1b9b4d#:~:text=Activation%20functions%20are%20mathematical%20functions,fired)%20based%20on%20its%20input`. Accessed: 2024-12-08.

[8] What are kernel initializers and their significance?, . URL `https://datascience.stackexchange.com/questions/37378/what-are-kernel-initializers-and-what-is-their-significance`. Accessed: 2024-12-08.

[9] Difference between a batch and an epoch. URL https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=The%20number%20of%20epochs%20is%20a%20hyperparameter%20that%20defines%20the,of%20one%20or%20more%20batches. Accessed: 2024-12-08.

[10] Impact of learning rate on a model. URL https://www.geeksforgeeks.org/impact-of-learning-rate-on-a-model/#:~:text=Learning%20rate%20is%20a%20hyperparameter,the%20loss%20function%20during%20optimization. Accessed: 2024-12-08.

[11] How can you use momentum to optimize neural networks? URL https://www.linkedin.com/advice/0/how-can-you-use-momentum-optimize-neural-cztnf#:~:text=Momentum%20is%20a%20technique%20used%20in%20optimizing%20neural%20networks%20to,step%20to%20the%20current%20update. Accessed: 2024-12-08.

[12] Batch normalization. URL https://viso.ai/deep-learning/batch-normalization/#:~:text=Batch%20normalization%20is%20a%20method,providing%20faster%20neural%20network%20convergence. Accessed: 2024-12-08.

[13] Flattening in neural network. URL https://medium.com/@vaibhav1403/flattening-in-neural-network-10e260d2b06f#:~:text=In%20the%20context%20of%20neural%20networks%2C%20especially%20convolutional%20neural%20networks,into%20a%20one-dimensional%20vector. Accessed: 2024-12-08.

[14] Stochastic gradient descent: A popular optimization method. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC10426722/#:~:text=SGD%20is%20a%20popular%20optimisation,with%20respect%20to%20each%20weight. Accessed: 2024-12-08.

[15] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[16] Cnn modeling with image translations using mnist data. URL https://fairyonice.github.io/CNN-modeling-with-image-translations-using-MNIST-data.html. Accessed: 2024-12-08.