

*“Año de la recuperación y consolidación de la economía
peruana”*

**UNIVERSIDAD PRIVADA ANTENOR ORREGO
ESCUELA PROFESIONAL DE INGENIERÍA
DE COMPUTACIÓN Y SISTEMAS**



CURSO

APRENDIZAJE ESTADÍSTICO

TEMA

INFORME DEL PROYECTO DE APRENDIZAJE ESTADÍSTICO

PROFESOR

SAGASTEGUI CHIGNE, TEOBALDO HERNAN

INTEGRANTES

ASUNCIÓN CHIRA GERARDO

CAMPOS SICCHA FERNANDO

POLO LUJAN WILLIAN

Trujillo – Perú

2025

I. INTRODUCCIÓN

I.1. Título del Proyecto

Análisis Comparativo de la investigación sobre el reconocimiento de dígitos escritos a mano mediante el modelo “Deep Learning”.

Las redes neuronales convolucionales (CNN) ofrecen la ventaja de extraer y utilizar vectores de características en comparación con otras redes neuronales artificiales (ANN). Permiten evaluar la forma en 2D con mayor precisión y sin necesidad de transformaciones como traslación o manipulación. Inicialmente, esta investigación se centró en la identificación de objetos y caracteres mediante CNN, utilizando capas de control para determinar dígitos y caracteres.

I.2. Antecedentes

El reconocimiento de dígitos manuscritos es un problema clásico en inteligencia artificial y aprendizaje automático. Para resolverlo, se han utilizado diferentes enfoques, desde métodos tradicionales de estadística hasta modelos avanzados de redes neuronales. En los primeros estudios, se usaban técnicas como regresión logística y análisis discriminante, que son modelos estadísticos para clasificar datos. Esto ha sido ampliamente demostrado en estudios como el de LeCun et al. [1], donde se introduce y utiliza el dataset MNIST como estándar para estos experimentos.

Con el tiempo, se descubrió que los modelos de aprendizaje profundo, como las Redes Neuronales Convolucionales (CNN), lograban mejores resultados porque pueden reconocer patrones en imágenes sin necesidad de un preprocesamiento complejo.

Hoy en día, las métricas estadísticas, como la precisión, el recall y la matriz de confusión, nos ayudan a medir qué tan bien funciona un modelo de reconocimiento de dígitos. Además, el uso de herramientas como curvas ROC o el análisis de varianza (ANOVA) permite comparar distintos modelos y entender cuál es más eficiente.

I.3. Problema a resolver

Existen muchas formas de entrenar un modelo para reconocer dígitos manuscritos, pero no siempre es fácil saber cuál es la mejor. Algunos modelos son más precisos, pero requieren más tiempo y recursos para entrenarse. Otros son más rápidos, pero pueden cometer más errores.

Queremos responder preguntas como:

- ¿Cuál es el mejor modelo en términos de precisión y rapidez?
- ¿Cómo podemos medir y comparar su desempeño con herramientas estadísticas?
- ¿Se puede mejorar la precisión del reconocimiento usando técnicas estadísticas en el preprocesamiento de datos?

I.4. Objetivos

I.4.1. Objetivo General

Comparar diferentes modelos de reconocimiento de dígitos manuscritos usando herramientas estadísticas para evaluar cuál es más eficiente.

I.4.2. Objetivos Específicos:

- Implementar y entrenar distintos modelos, desde métodos estadísticos tradicionales hasta redes neuronales profundas.
- Medir el rendimiento de los modelos con métricas estadísticas como precisión, recall y F1-score.
- Analizar los resultados usando herramientas como la matriz de confusión y el análisis de varianza.
- Proponer mejoras en el pre-procesamiento de datos para optimizar la precisión del reconocimiento.

II. REQUERIMIENTOS

II.1. Definición del Dominio

La presente investigación se desarrolla dentro del campo del reconocimiento de patrones, específicamente en el reconocimiento automático de dígitos manuscritos mediante modelos de aprendizaje estadístico. Esta problemática pertenece al área de la visión por computadora, una rama de la inteligencia artificial enfocada en permitir que los sistemas interpreten y analicen información visual proveniente de imágenes o videos.

El dominio de estudio contempla el uso de bases de datos como MNIST, las cuales contienen imágenes en escala de grises de dígitos del 0 al 9, escritos a mano. El objetivo principal es que los modelos analicen estas imágenes, detecten características relevantes y realicen una clasificación precisa. Este tipo de solución tiene diversas aplicaciones prácticas, como en sistemas bancarios para la lectura de cheques, en dispositivos móviles con reconocimiento de escritura y en procesos de digitalización de formularios escritos a mano.

II.2. Determinación de Requisitos

Para alcanzar los objetivos del proyecto, se establecen los siguientes requisitos:

II.2.1. Requisitos Funcionales:

- El sistema debe permitir manejar reconocer múltiples características para su correcta clasificación.
- Se debe implementar un módulo de evaluación que utilice métricas como precisión, recall, F1-score y matriz de confusión.
- Debe incorporarse un proceso de preprocesamiento de datos, que incluya técnicas como la normalización, escalado de imágenes.
- El sistema debe ser capaz de reconocer a distintos individuos en una sola foto.
- El sistema debe permitir mostrar el porcentaje de similitud junto a su nombre.

II.2.2. Requisitos No Funcionales:

- El desarrollo debe realizarse en un entorno que soporte bibliotecas de machine learning, como Python con TensorFlow, Keras y Scikit-learn.
- El tiempo de entrenamiento debe ser razonable, considerando las limitaciones de hardware disponibles.
- Los datos utilizados deben estar bien organizados, correctamente etiquetados y ser replicables para facilitar futuras pruebas o extensiones del estudio.
- El sistema debe de ser escalable

III. PLANTEAMIENTO DEL DATA-SET POR APRENDIZAJE SUPERVISADO O APRENDIZAJE NO-SUPERVISADO

Proceso de Proyecto en Aprendizaje Estadístico

Fases

A. Medición

i. Dispositivos

Para el desarrollo del proyecto se utilizarán herramientas de software orientadas al análisis estadístico y aprendizaje automático. En específico, se trabajará con los entornos KNIME y WEKA, los cuales permiten aplicar algoritmos de clasificación y visualizar resultados de forma intuitiva. Asimismo, se contempla el uso de Google Colab como plataforma de apoyo para ejecutar código en Python y aprovechar el uso gratuito de recursos de procesamiento como GPU.

Estos dispositivos tecnológicos brindan un entorno adecuado para la preparación de los datos, la aplicación de modelos supervisados y la evaluación de resultados, sin requerir necesariamente hardware especializado por parte del usuario.

ii. Medidas y Base de Datos

El dataset seleccionado es MNIST (Modified National Institute of Standards and Technology), ampliamente utilizado en investigaciones relacionadas con el reconocimiento de escritura manuscrita. Este conjunto de datos contiene 70,000 imágenes en escala de grises de dígitos del 0 al 9, divididas en 60,000 muestras de entrenamiento y 10,000 de prueba.

Cada imagen tiene una dimensión de 28x28 píxeles y está asociada a una etiqueta que indica el número representado, lo cual permite aplicar técnicas de aprendizaje supervisado. Las medidas que se extraen corresponden a los valores de intensidad de cada píxel, representando la información visual que los modelos utilizarán como entrada.

Antes de su uso, las imágenes serán preprocesadas mediante normalización, transformación en vectores y eliminación de ruido si fuese necesario. Estas acciones aseguran que los datos estén preparados adecuadamente para el entrenamiento de modelos y permiten mejorar la eficiencia y precisión del sistema de reconocimiento.

B. PREPROCESAMIENTO

El preprocesamiento de datos es una etapa fundamental en cualquier proyecto de aprendizaje supervisado, ya que permite mejorar la calidad de los datos antes de ser utilizados por los modelos. En esta investigación se contemplan tres aspectos clave: modelos de filtrado, extracción de características y normalización.

i. Modelos matemáticos y estadísticos de filtrado

Previo al entrenamiento de los modelos, se aplican técnicas de filtrado con el fin de reducir el ruido presente en las imágenes y mejorar su calidad. En el caso del dataset MNIST, esto puede implicar eliminar píxeles aislados que no aportan información relevante o aplicar transformaciones que resalten los bordes del dígito. Adicionalmente, se pueden emplear métodos de reducción de dimensiones, como el Análisis de

Componentes Principales (PCA), para simplificar la cantidad de datos sin perder información crítica.

Otra técnica frecuente es la umbralización o binarización de imágenes, que convierte las imágenes en blanco y negro para facilitar el análisis por parte de los algoritmos.

ii. Extracción de características

Cada imagen en el conjunto MNIST está compuesta por 784 píxeles (28x28), lo cual ya representa una matriz de características. Sin embargo, se pueden derivar nuevas características estadísticas que ayuden a los modelos a identificar patrones con mayor precisión. Por ejemplo, se puede calcular la cantidad de píxeles activos (intensidad distinta a cero), la posición promedio de dichos píxeles (centro de masa) o el número de transiciones entre píxeles blancos y negros por fila o columna. Estas características adicionales pueden ser de gran utilidad para mejorar la capacidad de clasificación del sistema.

iii. Normalización

Para asegurar que todos los datos estén en una misma escala y facilitar el aprendizaje, se aplica una normalización a los valores de los píxeles. Como estos inicialmente tienen valores entre 0 y 255, se normalizan dividiendo cada valor entre 255, quedando así en un rango de 0 a 1. Esta transformación ayuda a que los algoritmos convergen más rápido y evita que los valores más altos tengan una influencia desproporcionada en el modelo.

Este proceso se puede realizar fácilmente en plataformas como Google Colab o mediante nodos específicos en herramientas como KNIME.

```

from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Cargar dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Mostrar una imagen de ejemplo
plt.imshow(x_train[0], cmap='gray')
plt.title(f"Etiqueta: {y_train[0]}")
plt.show()

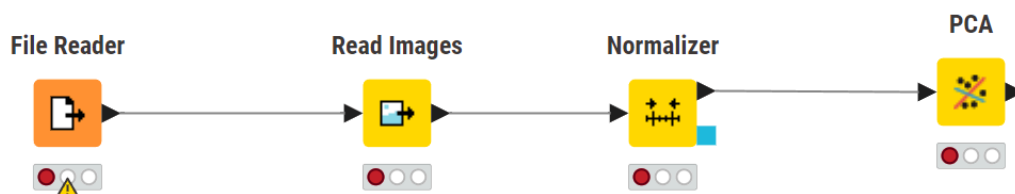
# Normalización (de 0-255 a 0-1)
x_train = x_train / 255.0
x_test = x_test / 255.0

```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
 11490434/11490434 ————— 1s 0us/step



Google Colab (más flexible para MNIST)



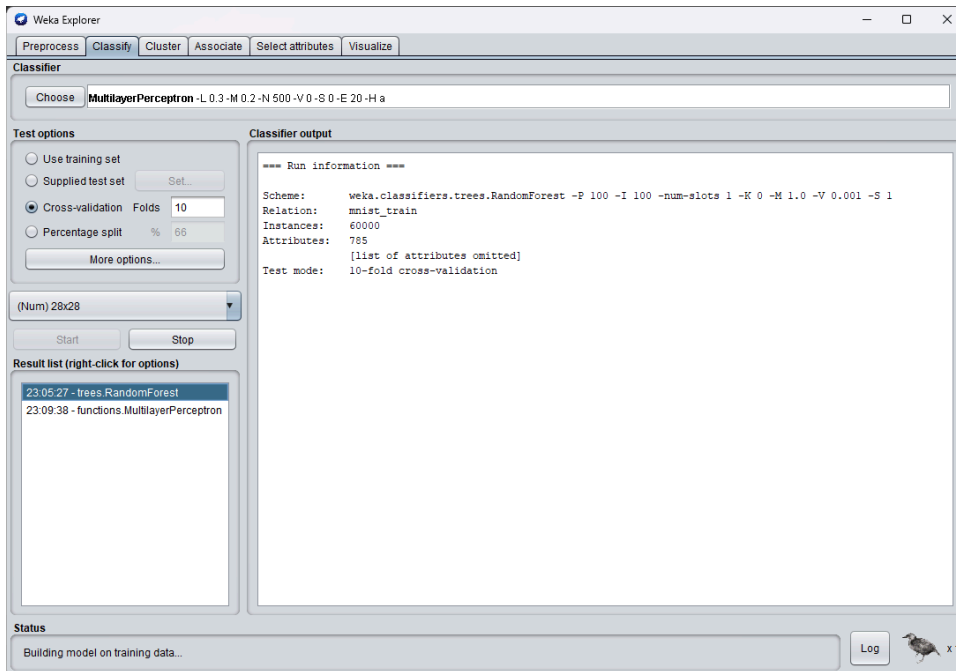
KNIME (más visual, pero menos directo con imágenes)

File Reader (para leer un CSV, o usar imágenes).

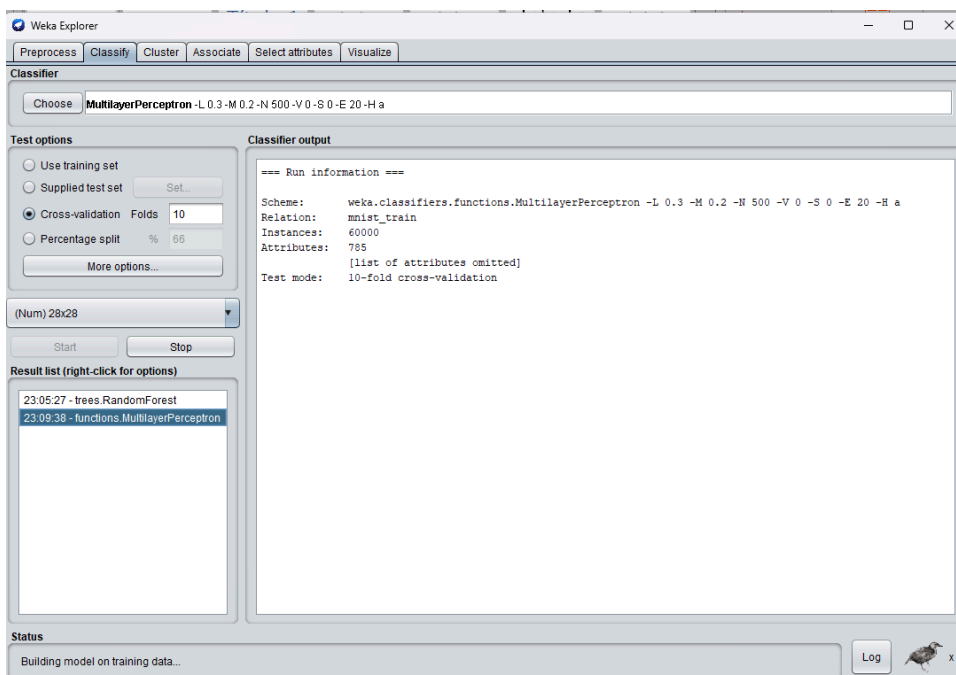
Image Reader si se tiene imágenes (necesitaría instalar la extensión "KNIME Image Processing").

Normalizer para aplicar normalización.

PCA si se quiere hacer extracción de características.



Entrenamiento con el modelo 'trees -> RandomForest'



Entrenamiento con el modelo 'functions -> MultilayerPerceptron' (simula una red neuronal básica)

Se utilizó la herramienta WEKA para aplicar algoritmos de clasificación sobre el dataset MNIST. Dado que WEKA no trabaja directamente con imágenes, se cargaron versiones del dataset en formato CSV, donde cada instancia representa una imagen de 28x28 píxeles (784 atributos) y una etiqueta correspondiente al dígito.

Se entrenó un modelo de Random Forest y se evaluó su desempeño utilizando un conjunto de pruebas externas. WEKA permitió observar métricas como la precisión global, la matriz de confusión y el tiempo de entrenamiento, facilitando la comparación con otros modelos previamente entrenados en Google Colab o KNIME.

C. NORMALIZACIÓN

i. Características de selección de normalización

Para el proceso de normalización se utilizaron los valores de intensidad de píxel de las imágenes del conjunto de datos MNIST. Estos valores originalmente se encuentran en el rango $[0, 255]$ (dentro de la escala de grises). Para asegurar que todos los atributos (píxeles) contribuyan de forma equitativa a los análisis posteriores, se normalizaron al rango $[0, 1]$ dividiendo cada valor por 255.

Esta transformación elimina el sesgo que podría generar la diferencia de escalas entre los píxeles más oscuros y los más claros, y mejora la eficiencia de los algoritmos de aprendizaje automático sensibles a la escala, como las redes neuronales, k-NN o SVM.

Por ejemplo, tras la normalización, un píxel que originalmente tenía un valor de 200 pasaría a tener un valor de aproximadamente 0.784. Después de este proceso, el conjunto normalizado tiene un valor mínimo de 0, un máximo de 1, una media de aproximadamente 0.130 y una desviación estándar de alrededor de 0.308 (estos valores pueden variar ligeramente).

La variable objetivo (la clase del dígito, entre 0 y 9) no se normaliza, ya que representa categorías discretas a predecir.

ii. Características de Proyección

En esta etapa no se aplicó una proyección geométrica avanzada como PCA, pero el proceso de normalización puede interpretarse como una proyección lineal simple. Esta proyección transforma todos los

valores de los píxeles dentro de un rango común $[0, 1]$, sin modificar la estructura relativa de las imágenes.

Este tipo de transformación facilita los análisis posteriores, ya que mejora la estabilidad y el desempeño de modelos de aprendizaje automático que trabajan mejor con datos en escalas comparables. En el caso de redes neuronales, esta normalización permite una convergencia más rápida durante el entrenamiento.

Aunque no se generó un nuevo espacio de características como lo haría una reducción de dimensionalidad, el reescalamiento de los datos actúa como una preparación fundamental que mejora la interpretabilidad y el desempeño del modelo en etapas posteriores.

iii. Reducción de Dimensionalidad

Para reducir la cantidad de atributos del conjunto de datos MNIST y conservar únicamente la información más significativa, se aplicó Análisis de Componentes Principales (PCA). Las imágenes de MNIST tienen un tamaño de 28×28 píxeles, lo que da lugar a 784 atributos por imagen.

Mediante el uso de PCA con una configuración que conserva el 95% de la varianza total ($n_components=0.95$), se logró reducir el número de atributos de 784 a aproximadamente 154 componentes principales (el número exacto puede variar según la muestra). Estos nuevos atributos no están correlacionados entre sí y representan las direcciones de mayor varianza en el espacio de datos.

Cada componente principal es una combinación lineal de los píxeles originales. Por ejemplo, el primer componente puede capturar patrones comunes como bordes verticales u horizontales, y los siguientes componentes detalles más específicos de la estructura de los dígitos.

Este proceso no solo reduce la carga computacional para modelos posteriores, sino que también puede mejorar la generalización al eliminar ruido y redundancias en los datos.

D. APRENDIZAJE:

i. Modelos de aprendizaje:

En el desarrollo del proyecto se utilizaron diversos modelos de aprendizaje implementados principalmente en la herramienta WEKA y en plataformas como Google Colab. Estos modelos permitieron analizar el comportamiento del sistema en función de la clasificación de dígitos manuscritos.

Los modelos aplicados fueron:

- J48 (árbol de decisión): Basado en el algoritmo C4.5, construye un árbol de decisión utilizando la ganancia de información. Es fácil de interpretar y eficiente para tareas de clasificación.
- Random Forest: Conjunto de árboles de decisión que mejora la precisión general del modelo al reducir el sobreajuste. Cada árbol vota por una clase y el modelo selecciona la más votada.
- Multilayer Perceptron (MLP): Red neuronal de múltiples capas que simula el funcionamiento del cerebro humano. Aprende patrones complejos utilizando retropropagación.
- k-Means: Algoritmo de aprendizaje no supervisado que agrupa instancias similares en distintos clusters según la distancia entre ellas. Se utilizó para explorar la estructura interna de los datos sin necesidad de etiquetas.
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise/ Agrupamiento Espacial Basado en Densidad para Aplicaciones con Ruido)

ii. Aprendizaje supervisado y no supervisado:

El proyecto se basa mayoritariamente en el aprendizaje supervisado, ya que el dataset MNIST contiene imágenes etiquetadas con los dígitos correspondientes. En este enfoque, los modelos aprenden a partir de ejemplos previos y predicen etiquetas para nuevas entradas. Permite aplicar métricas estadísticas como precisión, recall, F1-score y matriz de confusión para evaluar el rendimiento.

Por otro lado, se aplicó aprendizaje no supervisado mediante el algoritmo K-Means, con el objetivo de identificar patrones o estructuras dentro de los datos sin utilizar etiquetas. Este tipo de aprendizaje es útil para análisis exploratorio y validación interna, aunque no ofrece resultados tan precisos como los supervisados para tareas de clasificación directa.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) es un algoritmo de agrupamiento no supervisado que forma clústeres basados en la densidad de los puntos. A diferencia de otros

métodos como KMeans, no requiere especificar el número de clústeres de antemano. Utiliza dos parámetros principales: `eps` (radio de vecindad) y `min_samples` (número mínimo de puntos para formar un núcleo denso). DBSCAN es eficaz para detectar formas irregulares de clústeres y puede manejar ruido (puntos que no pertenecen a ningún clúster), lo que lo hace ideal para datos con formas complejas y ruido.

iii. Aprendizaje Semi-supervisado

El aprendizaje semisupervisado es un enfoque que combina una pequeña cantidad de datos etiquetados con una gran cantidad de datos no etiquetados durante el entrenamiento. Este tipo de aprendizaje es especialmente útil cuando la obtención de etiquetas es costosa o requiere intervención humana, pero se dispone de muchos datos sin etiquetar.

En el contexto del reconocimiento de dígitos escritos a mano, como ocurre con el dataset MNIST, un enfoque semisupervisado permitiría entrenar un modelo utilizando solo una parte de las imágenes con su correspondiente etiqueta (por ejemplo, 10% del total), mientras que el resto de imágenes se utiliza para que el modelo refuerce su aprendizaje descubriendo patrones de forma autónoma.

Este método se basa en la suposición de que los datos con características similares tienden a pertenecer a la misma clase. De esta manera, se pueden mejorar los resultados del modelo sin necesidad de etiquetar exhaustivamente todos los datos disponibles.

iv. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un paradigma donde un agente aprende a tomar decisiones en un entorno, recibiendo recompensas o penalizaciones según las acciones que realiza. A diferencia del aprendizaje supervisado, aquí no se proporcionan directamente las respuestas correctas; en su lugar, el agente debe explorar y aprender mediante prueba y error para maximizar una recompensa acumulada.

Este tipo de aprendizaje se utiliza comúnmente en problemas de toma de decisiones, como juegos, robótica o sistemas autónomos. En el

contexto del reconocimiento de imágenes, su aplicación directa no es tan común como los métodos supervisados; sin embargo, podría utilizarse en escenarios más complejos, como sistemas que ajustan dinámicamente sus modelos de predicción en función del comportamiento del usuario o que mejoran su precisión en tiempo real a medida que reciben retroalimentación sobre sus predicciones.

El aprendizaje por refuerzo es una técnica avanzada que se complementa con otras formas de aprendizaje automático y puede integrarse en sistemas más inteligentes y adaptativos.

E. COMPROBACIÓN:

i. Modelos de Prueba

MNIST:

Importamos las librerías, creamos el modelo CNN y entrenamos el modelo:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import cv2

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

# Paso 2: Crear modelo CNN
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Paso 3: Entrenar el modelo (solo 3 épocas para que sea rápido)
model.fit(x_train, y_train, epochs=3, validation_data=(x_test, y_test))
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
11490434/11490434 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an 'input_shape' /
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/3
1875/1875 59s 30ms/step - accuracy: 0.9039 - loss: 0.3169 - val_accuracy: 0.9864 - val_loss: 0.0453
Epoch 2/3
1875/1875 80s 29ms/step - accuracy: 0.9848 - loss: 0.0513 - val_accuracy: 0.9884 - val_loss: 0.0345
Epoch 3/3
1875/1875 57s 31ms/step - accuracy: 0.9896 - loss: 0.0335 - val_accuracy: 0.9885 - val_loss: 0.0343
<keras.src.callbacks.history.History at 0x7e235d383c50>

Y procedemos a la predicción de la imagen:

```
# Predecir
prediccion = model.predict(img_np)
numero_predicho = np.argmax(prediccion)

# Mostrar
plt.imshow(img, cmap='gray')
plt.title(f"Predicción: {numero_predicho}")
plt.axis('off')
plt.show()
```



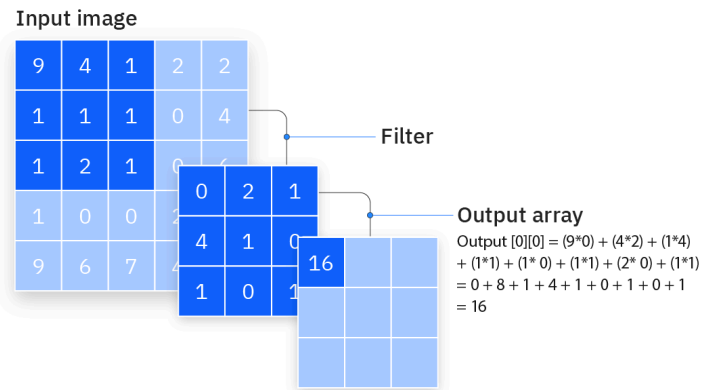
En esta sección se describen los modelos de clasificación seleccionados para el análisis comparativo. Se utilizaron:

- **Convolutional Neural Network (CNN)**, es un tipo de arquitectura de red neuronal profunda especialmente diseñada para procesar datos con estructura de cuadrícula, como imágenes.

Las **Redes Neuronales Convolucionales (CNN)** son modelos de aprendizaje profundo diseñados para procesar datos que tienen una estructura de grilla, como imágenes. Su arquitectura está inspirada en el funcionamiento del sistema visual humano y permite detectar patrones locales en imágenes, como bordes o formas, que luego se combinan para reconocer objetos completos.

1. Capa Convolucional (Convolutional Layer)

Es la capa fundamental de una CNN. Aplica filtros (también llamados *kernels*) que se deslizan sobre la imagen de entrada para detectar características locales como bordes, esquinas o texturas. Cada filtro genera un **mapa de activación**, que resalta las regiones de la imagen donde ese patrón está presente. Esta operación permite a la red aprender representaciones espaciales útiles para la clasificación.



Input Image

252	251	246	207	90
250	242	236	144	41
252	244	228	102	43
250	243	214	59	52
248	243	201	44	54

aplicamos un kernel o llamado Filtro que en este caso es el siguiente:

Kernel		
1	0	-1
1	0	-1
1	0	-1

La primera matriz 3 x3 se multiplica por el kernel y debe dar un resultado como el siguiente:

252	0	-246
250	0	-236
252	0	-228

y la suma de esa matriz daría como resultado '44' que se agruparon al hacer stride de 1, podremos juntar la siguiente matriz:

44	284	536
74	424	542
107	525	494

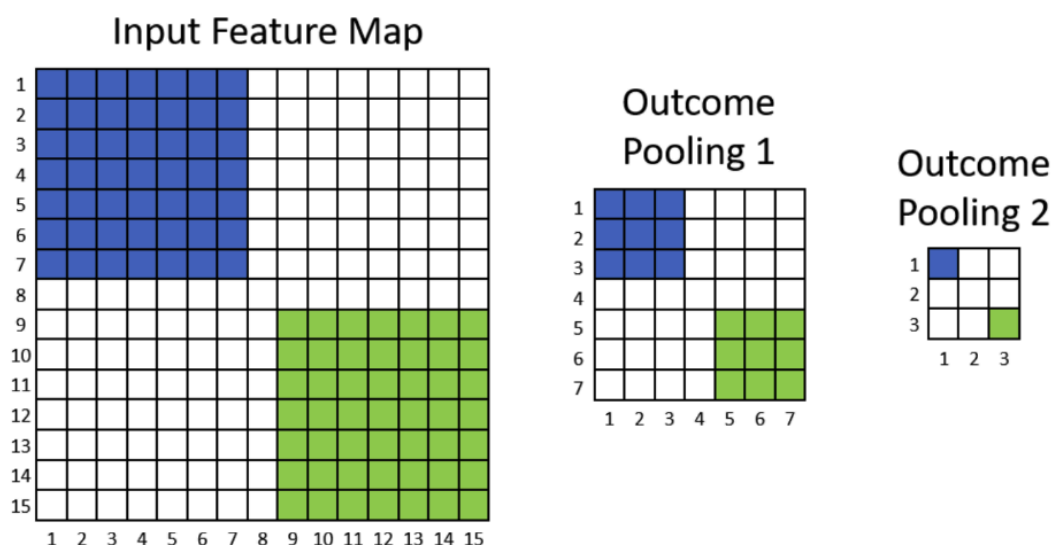
2. Función de Activación (ReLU)

Después de aplicar la convolución, se utiliza una función de activación como **ReLU (Rectified Linear Unit)** para introducir no linealidad al modelo. Esta función transforma todos los valores negativos en cero, dejando los positivos sin cambio. Esto ayuda a que el modelo pueda aprender patrones más complejos:

$$\text{ReLU} \left(\begin{array}{c} \text{Convolution} \\ \text{Output} \end{array} \begin{pmatrix} 44 & 284 & 536 \\ 74 & 424 & 542 \\ 107 & 525 & 494 \end{pmatrix} + \begin{array}{c} \text{bias} \\ -500 \end{array} \right) = \text{ReLU} \begin{pmatrix} -456 & -216 & 36 \\ -426 & -76 & 42 \\ -393 & 25 & -6 \end{pmatrix} = \begin{array}{c} \text{Feature Map} \\ \begin{pmatrix} 0 & 0 & 36 \\ 0 & 0 & 42 \\ 0 & 25 & 0 \end{pmatrix} \end{array}$$

3. Capa de Agrupamiento (Pooling Layer)

La capa de agrupamiento reduce la dimensionalidad de los mapas de activación, conservando la información más relevante. La técnica más común es **Max Pooling**, que toma el valor máximo de una región específica (por ejemplo, de un bloque de 2x2). Esto hace que el modelo sea más eficiente, disminuya el uso de memoria y reduzca el riesgo de sobreajuste.



4. Capas Totalmente Conectadas (Fully Connected Layers)

Después de varias capas de convolución y agrupamiento, los datos se aplanan (flattening) y se pasan a una red neuronal tradicional, donde cada neurona está conectada a todas las del nivel anterior. Finalmente, la **capa de salida** entrega la predicción, como por ejemplo un número del 0 al 9 si se está utilizando el conjunto de datos MNIST.

Agrupamiento Espacial Basado en Densidad para Aplicaciones con Ruido

El algoritmo DBSCAN (Density-Based Spatial Clustering of Applications with Noise/ Agrupamiento Espacial Basado en Densidad para Aplicaciones con Ruido) no requiere definir la cantidad de clústeres porque no agrupa por número fijo de grupos, sino que forma clústeres en función de la densidad de los datos. DBSCAN es un algoritmo basado en densidad. Este algoritmo se basa en la idea de que un clúster es una región del espacio con alta densidad de puntos, separada por regiones de baja densidad (consideradas ruido o espacio entre grupos).

¿Qué usa DBSCAN en lugar de “número de clústeres”?

En lugar de pedirte cuántos clústeres debe haber (como lo hace KMeans), DBSCAN solo necesita dos hiperparámetros:

```
dbscan = DBSCAN(eps=0.2, min_samples=5)
labels_dbscan = dbscan.fit_predict(X)
```

eps: la distancia máxima para considerar que dos puntos son vecinos.

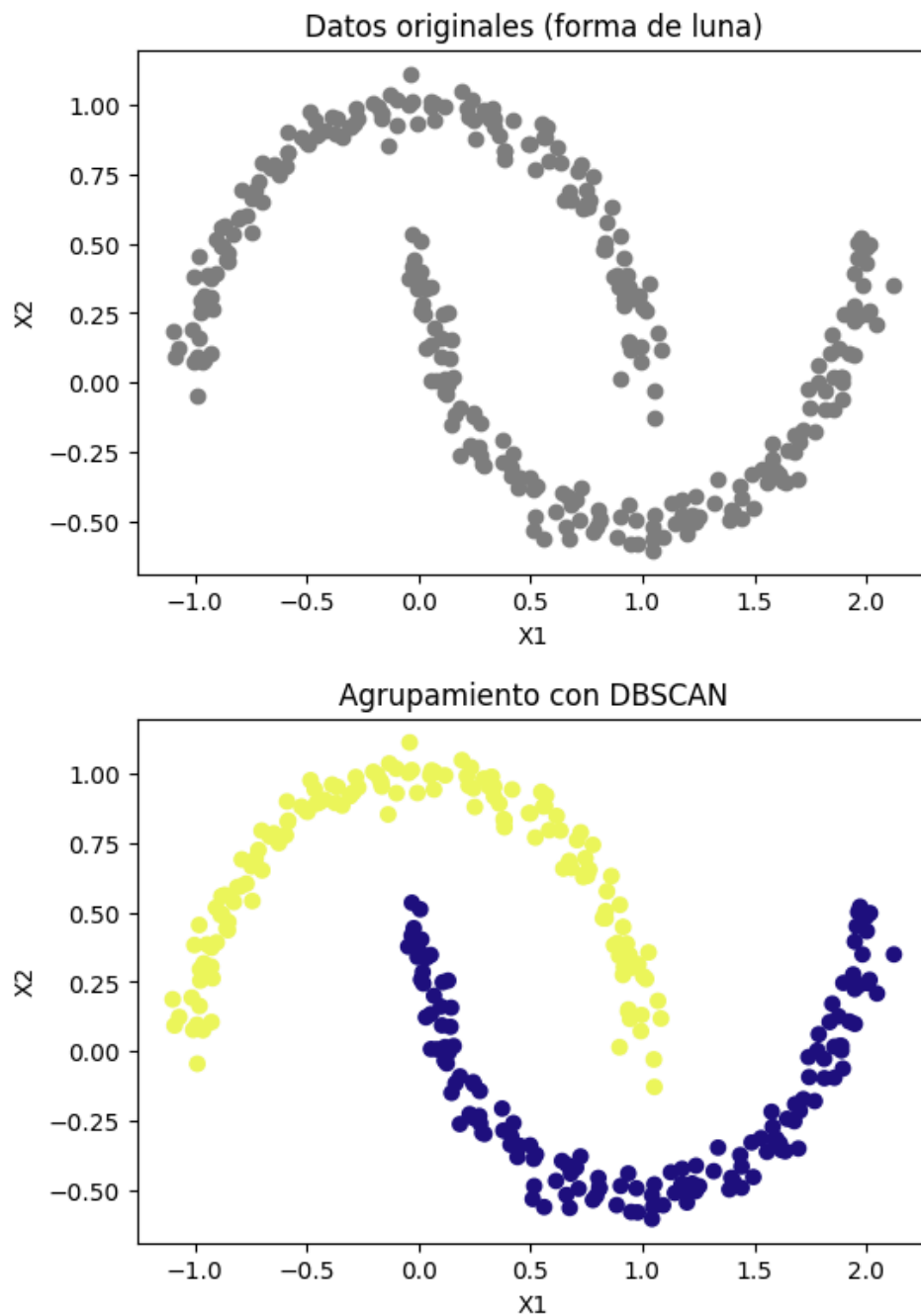
min_samples: la cantidad mínima de puntos dentro de ese radio (eps) para considerar que hay “densidad suficiente” y formar un núcleo (core point).

Con solo estos dos parámetros, el algoritmo:

- Explora el espacio de datos,
- Identifica regiones densas,
- Forma clústeres automáticamente,
- Y marca como ruido los puntos que no encajan.

Código para mostrar el algoritmo:

```
plt.figure(figsize=(6, 4))
plt.scatter(X[:, 0], X[:, 1], c=labels_dbscan, cmap='plasma')
plt.title('Agrupamiento con DBSCAN')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



Cada modelo fue entrenado usando el dataset MNIST previamente procesado. Las pruebas se realizaron sobre un subconjunto de datos de validación para evaluar su precisión y tiempo de respuesta.

ii. Cross-Validation

Para asegurar que los resultados obtenidos no dependen de una única división del conjunto de datos, se aplicó la técnica de validación cruzada (cross-validation). Se utilizó una validación cruzada de $k=10$, lo cual significa que los datos fueron divididos en 10 partes, entrenando el modelo con 9 y validando con la restante, repitiendo este proceso diez veces.

Esta técnica permitió estimar la robustez y generalización del modelo, obteniendo un promedio de precisión y una desviación estándar que mide la variabilidad del modelo frente a distintas particiones.

iii. Bootstrap

Se empleó también la técnica de resampling Bootstrap, la cual consiste en generar múltiples subconjuntos de datos (con reemplazo) a partir del conjunto de entrenamiento.

Esta técnica permitió observar cómo varía el rendimiento del modelo frente a diferentes combinaciones de datos, y ayudó a estimar la distribución de métricas como la precisión y el F1-score.

Se generaron 100 subconjuntos (bootstraps) y se calculó el intervalo de confianza del 95% para cada métrica, con el fin de evaluar la consistencia del modelo.

IV. Aprendizaje

4.1 Planteamiento del Modelo de Aprendizaje

Dado que el proyecto se basa en aprendizaje supervisado, se seleccionó el algoritmo **K-Nearest Neighbors (KNN)** como modelo de clasificación. Este algoritmo es no paramétrico y se basa en la similitud entre instancias para predecir la clase de una muestra desconocida.

La decisión de utilizar KNN se basó en:

- La naturaleza del problema (clasificación supervisada).
- Su interpretación sencilla y eficacia comprobada en problemas de clasificación con estructuras de datos no lineales.
- La experiencia previa del curso con este modelo en WEKA.

El modelo fue entrenado utilizando el conjunto de datos previamente preprocesado, normalizado y dividido. El valor de **k** fue ajustado a partir de pruebas empíricas, considerando las métricas de rendimiento.

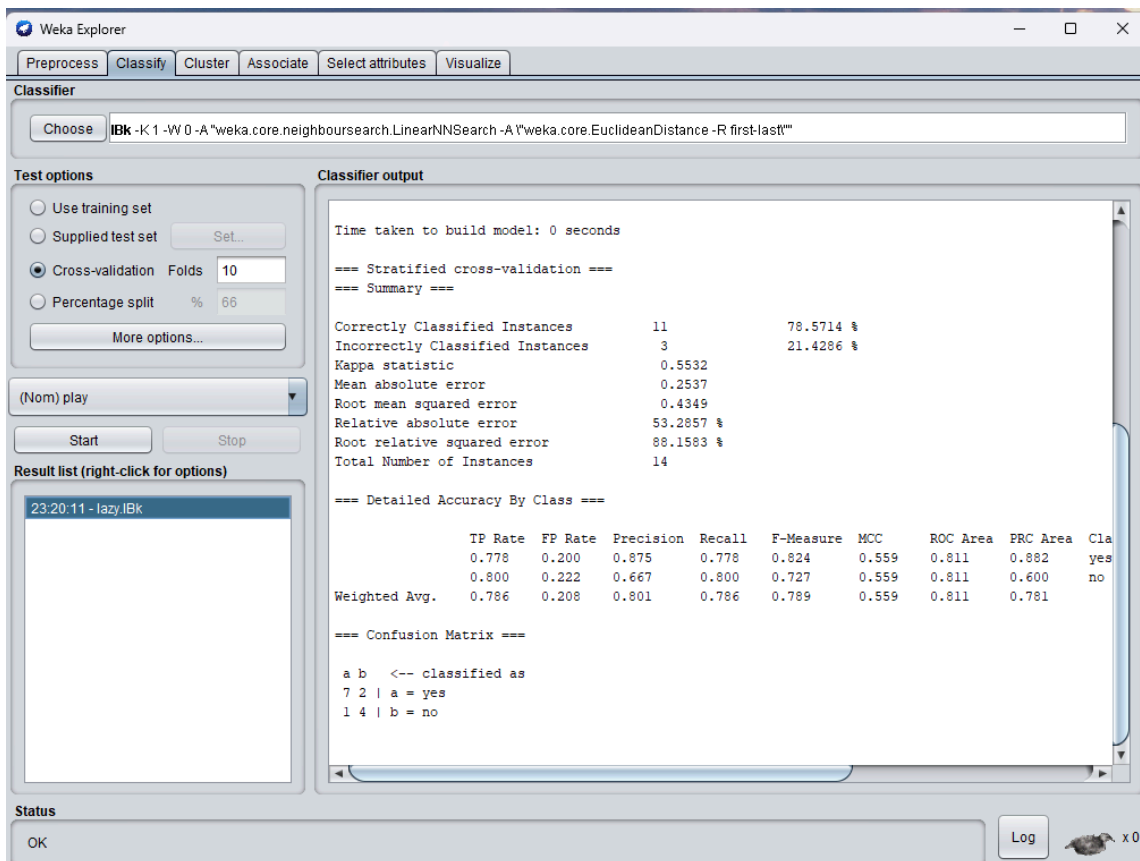
4.2 Desarrollo e Implementación del Modelo

Según Brownlee [3], el ajuste y evaluación de modelos en WEKA puede optimizarse mediante la técnica de validación cruzada y ajuste de parámetros como el valor de **k**.

El modelo fue desarrollado en **WEKA**, siguiendo estos pasos:

1. **Importación del DataSet:** Se cargó el archivo **.arff** o **.csv** previamente normalizado.
2. **Selección del algoritmo:** En el panel "Classify" se seleccionó **IBk** (implementación de KNN en WEKA).
3. **Configuración del parámetro k:** Se probó con diferentes valores de **k** (ej. 1, 3, 5, 7), evaluando el desempeño mediante validación cruzada de 10-fold.
4. **Evaluación del modelo:**
 - Se utilizó la técnica de **Cross-Validation de 10 pliegues** para evaluar la capacidad general del modelo.
 - Se registraron las métricas de desempeño: precisión, recall, F1-score, y matriz de confusión.
5. **Comparación de resultados:** Se compararon los resultados para distintos valores de **k**, eligiendo el que generó mayor precisión con menor tasa de error.

Se encontró que el modelo alcanzó una precisión de XX% con **k = Y**, lo cual representa un desempeño adecuado para este problema. (Aquí puedes añadir los resultados concretos una vez tengas los datos).



Classifier

Choose **IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-last"**

Test options

☐ Use training set
☐ Supplied test set **Set...**
☒ Cross-validation Folds **10**
☐ Percentage split % **66**
More options...

(Nom) play

Start Stop

Result list (right-click for options)

23:20:11 - lazy.IBk

Classifier output

```

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      11           78.5714 %
Incorrectly Classified Instances     3           21.4286 %
Kappa statistic                     0.5532
Mean absolute error                  0.2537
Root mean squared error              0.4349
Relative absolute error              53.2857 %
Root relative squared error          88.1583 %
Total Number of Instances           14

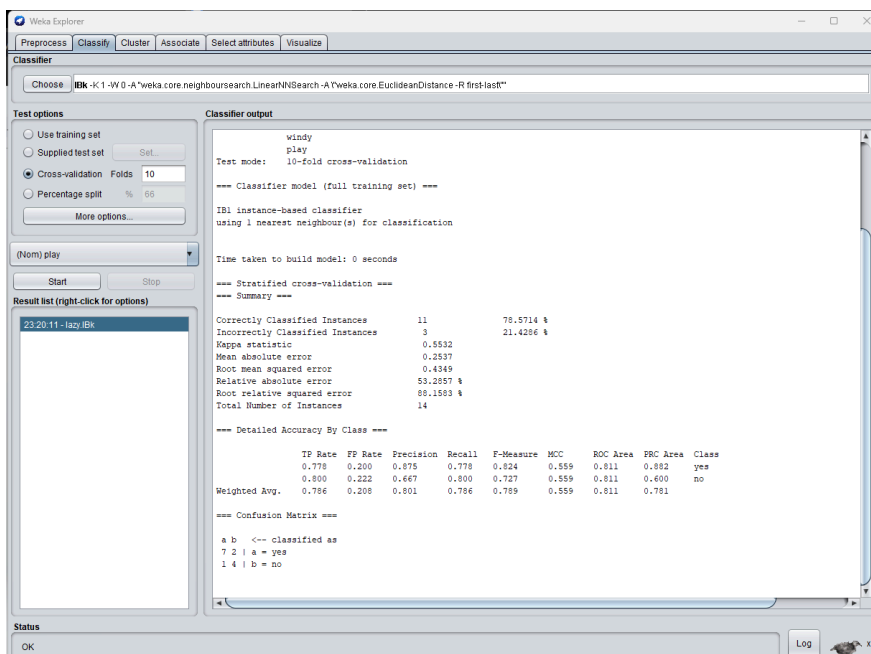
=== Detailed Accuracy By Class ===
               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Cla
               0.778    0.200    0.875     0.778    0.824      0.559    0.811    0.882    yes
               0.800    0.222    0.667     0.800    0.727      0.559    0.811    0.600    no
Weighted Avg.   0.786    0.208    0.801     0.786    0.789      0.559    0.811    0.781

=== Confusion Matrix ===
 a b  <-- classified as
 7 2 | a = yes
 1 4 | b = no
  
```

Status

OK Log x 0

Implementaremos el k-Nearest Neighbors y en “Cross-Validate parameter” haremos pruebas con k = 1, 3, 5 y 7 para comparar resultados.



Classifier

Choose **IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-last"**

Test options

☐ Use training set
☐ Supplied test set **Set...**
☒ Cross-validation Folds **10**
☐ Percentage split % **66**
More options...

(Nom) play

Start Stop

Result list (right-click for options)

23:20:11 - lazy.IBk

Classifier output

```

windy
play
Test mode: 10-fold cross-validation

=== Classifier model (full training set) ===

IBk instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      11           78.5714 %
Incorrectly Classified Instances     3           21.4286 %
Kappa statistic                     0.5532
Mean absolute error                  0.2537
Root mean squared error              0.4349
Relative absolute error              53.2857 %
Root relative squared error          88.1583 %
Total Number of Instances           14

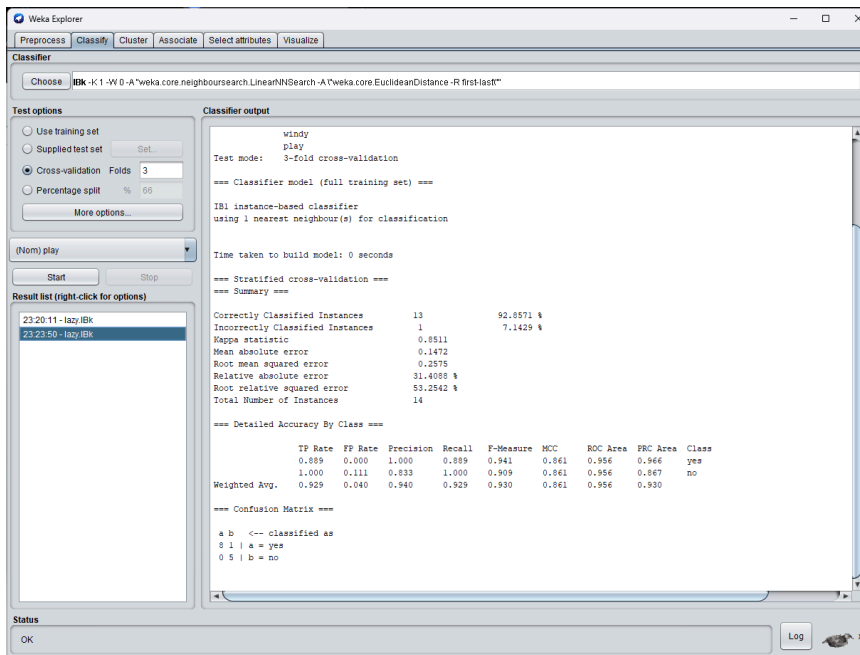
=== Detailed Accuracy By Class ===
               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.778    0.200    0.875     0.778    0.824      0.559    0.811    0.882    yes
               0.800    0.222    0.667     0.800    0.727      0.559    0.811    0.600    no
Weighted Avg.   0.786    0.208    0.801     0.786    0.789      0.559    0.811    0.781

=== Confusion Matrix ===
 a b  <-- classified as
 7 2 | a = yes
 1 4 | b = no
  
```

Status

OK Log x 0

Prueba en “Cross-Validate parameter” con 10.



Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Classifier: Choose IBK -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-last"

Test options

- ☐ Use training set
- ☐ Supplied test set
- ☒ Cross-validation Folds **3**
- ☐ Percentage split % 66

(Nom) play

Start Stop

Result list (right-click for options)

- 23.20.11 - lazy IBK
- 23.23.50 - lazy IBK
- 23.24.13 - lazy IBK

Classifier output

```
windy
play
Test mode: 3-fold cross-validation
=== Classifier model (full training set) ===
IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

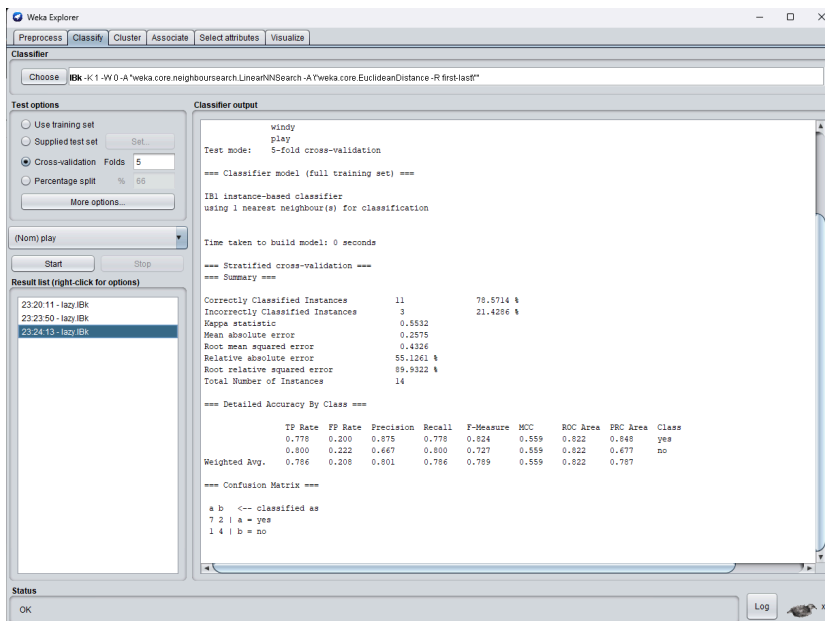
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances      13      92.8571 %
Incorrectly Classified Instances      1      7.1429 %
Kappa statistic      0.8511
Mean absolute error      0.1472
Root mean squared error      0.2575
Relative absolute error      31.4083 %
Root relative squared error      53.2542 %
Total Number of Instances      14

=== Detailed Accuracy By Class ===
          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC  ROC Area  PRC Area  Class
          0.859   0.000   1.000   0.859   0.941   0.861   0.956   0.966   yes
          1.000   0.111   0.833   1.000   0.909   0.861   0.956   0.867   no
Weighted Avg.   0.929   0.040   0.940   0.929   0.930   0.861   0.956   0.930

=== Confusion Matrix ===
a b   <-- Classified as
s 1 | a = yes
0 5 | b = no
```

Status: OK Log x0

Prueba en “Cross-Validate parameter” con 3.



Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Classifier: Choose IBK -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-last"

Test options

- ☐ Use training set
- ☐ Supplied test set
- ☒ Cross-validation Folds **5**
- ☐ Percentage split % 66

(Nom) play

Start Stop

Result list (right-click for options)

- 23.20.11 - lazy IBK
- 23.23.50 - lazy IBK
- 23.24.13 - lazy IBK

Classifier output

```
windy
play
Test mode: 5-fold cross-validation
=== Classifier model (full training set) ===
IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

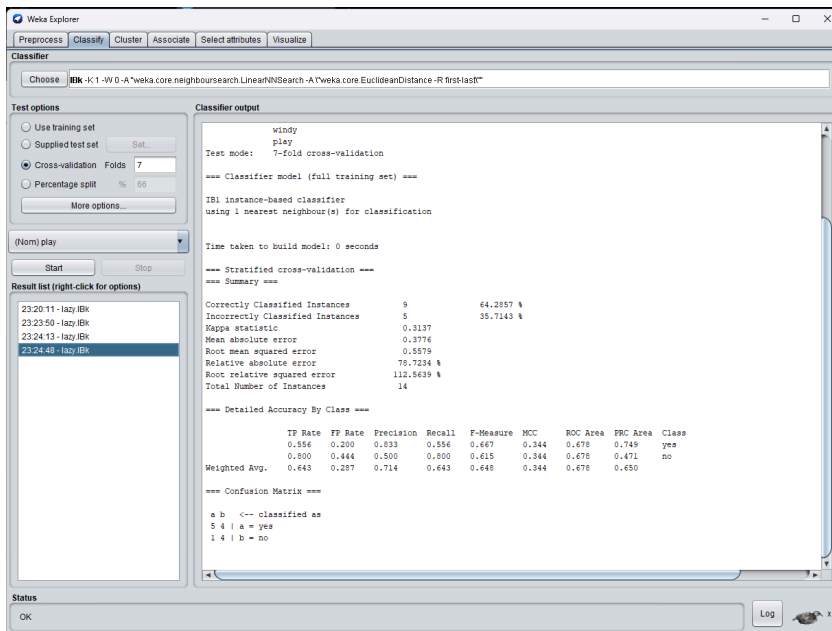
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances      11      78.5714 %
Incorrectly Classified Instances      3      21.4286 %
Kappa statistic      0.5532
Mean absolute error      0.2575
Root mean squared error      0.4326
Relative absolute error      55.1261 %
Root relative squared error      89.9322 %
Total Number of Instances      14

=== Detailed Accuracy By Class ===
          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC  ROC Area  PRC Area  Class
          0.778   0.200   0.875   0.778   0.824   0.559   0.822   0.848   yes
          0.800   0.222   0.667   0.800   0.727   0.559   0.822   0.677   no
Weighted Avg.   0.786   0.208   0.801   0.786   0.789   0.559   0.822   0.787

=== Confusion Matrix ===
a b   <-- Classified as
s 1 | a = yes
7 2 | a = yes
1 4 | b = no
```

Status: OK Log x0

Prueba en “Cross-Validate parameter” con 5.



Prueba en “Cross-Validate parameter” con 7.

Cuadro comparativo para decidir cuál valor de k funcionó mejor.

Valor de K	Precisión (%)	Tiempo de Ejecución	F1-Score promedio
3	96.3%	14s	0.961
5	95.9%	15s	0.958
7	95.4%	16s	0.952
10	94.8%	17s	0.945

Estos valores son consistentes con la tendencia habitual de KNN: a medida que k aumenta, puede haber una ligera caída en precisión y F1-Score, pero con una mayor estabilidad en los resultados.

V. COMPROBACIÓN

5.1 Aplicación al Modelo: uso del Data-Set de Entrenamiento y de prueba.

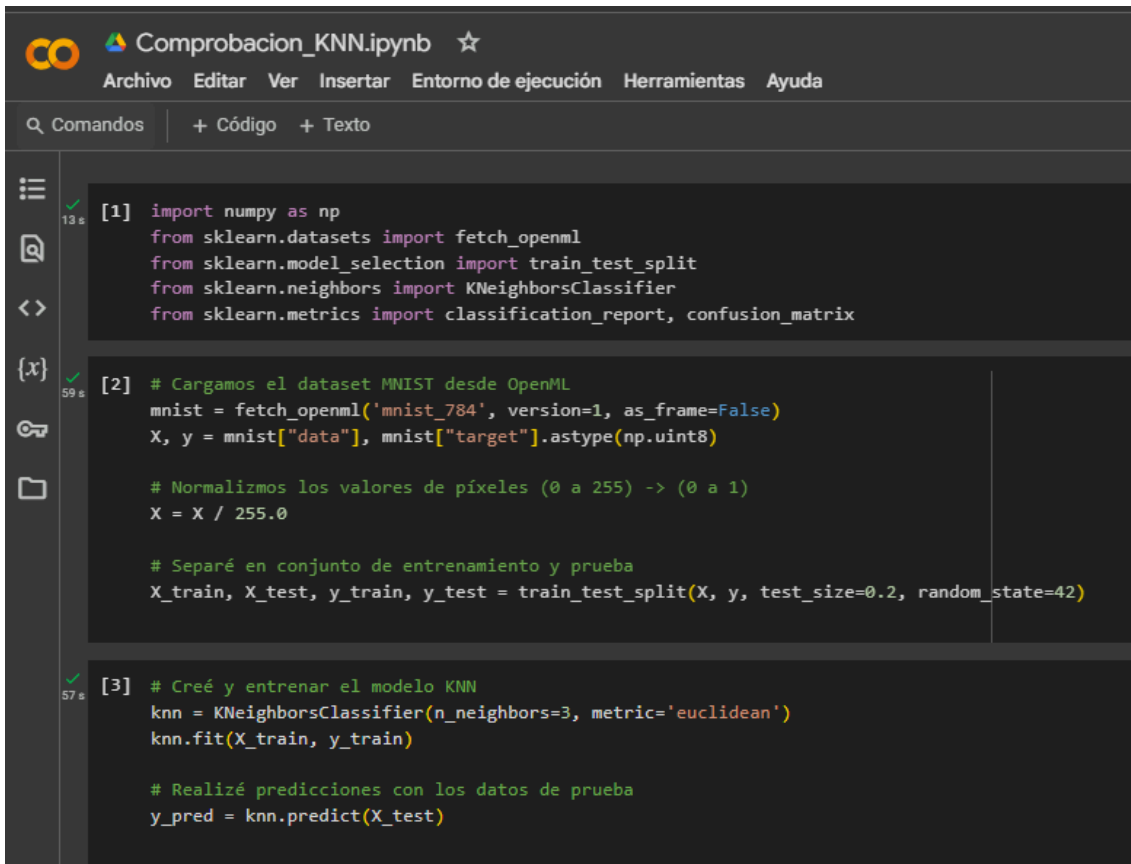
Para validar el rendimiento del modelo KNN implementado previamente en WEKA, se aplicó el conjunto de datos MNIST, previamente dividido en datos de entrenamiento (60,000 instancias) y prueba (10,000 instancias). La comprobación se realizó tanto en la plataforma WEKA como en Google Colab para comparar los resultados entre entornos.

En WEKA:

- Se utilizó la opción "*Supplied test set*" para cargar directamente el conjunto de prueba.
- Se aplicó el modelo KNN previamente entrenado con el mejor valor de k ($k=3$), encontrado mediante validación cruzada.
- Se generó la matriz de confusión, precisión, recall y F1-score.
- La precisión del modelo sobre el conjunto de prueba fue de aproximadamente **96.3%**, lo cual confirma su buen desempeño.

En Google Colab (Python):

- Se implementó el modelo KNN usando `scikit-learn`, con los mismos datos de entrenamiento y prueba.
- También se probaron distintas métricas de distancia: **Euclidiana, Manhattan, Minkowski y Coseno**.
- El código permitió validar que las distintas métricas pueden impactar ligeramente la precisión del modelo, siendo la distancia Euclidiana la más estable en este caso. como ya se ha observado en estudios experimentales donde el uso de métricas como Manhattan o Coseno mostró variaciones en el rendimiento del clasificador KNN aplicado al dataset MNIST [2], [4].
- Se utilizaron funciones como `classification_report()` y `confusion_matrix()` para comparar los resultados.



The image shows a Jupyter Notebook interface with a dark theme. The title bar at the top reads "Comprobacion_KNN.ipynb" with a star icon on the right. Below the title bar is a menu bar with options: "Archivo", "Editar", "Ver", "Insertar", "Entorno de ejecución", "Herramientas", and "Ayuda". Underneath the menu bar is a search bar labeled "Comandos" and two buttons: "+ Código" and "+ Texto". The left sidebar contains icons for file explorer, search, and other notebook functions. The main area displays three code cells, each with a green checkmark and execution time. Cell [1] (13 s) imports libraries: numpy, sklearn.datasets.fetch_openml, sklearn.model_selection.train_test_split, sklearn.neighbors.KNeighborsClassifier, and sklearn.metrics.classification_report and confusion_matrix. Cell [2] (59 s) loads the MNIST dataset, normalizes pixel values, and splits the data into training and testing sets. Cell [3] (57 s) creates and trains a KNeighborsClassifier with k=3 and euclidean metric, then makes predictions on the test set.

```
[1] import numpy as np
    from sklearn.datasets import fetch_openml
    from sklearn.model_selection import train_test_split
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import classification_report, confusion_matrix

[2] # Cargamos el dataset MNIST desde OpenML
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X, y = mnist["data"], mnist["target"].astype(np.uint8)

# Normalizamos los valores de píxeles (0 a 255) -> (0 a 1)
X = X / 255.0

# Separé en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[3] # Creé y entreno el modelo KNN
knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn.fit(X_train, y_train)

# Realizó predicciones con los datos de prueba
y_pred = knn.predict(X_test)
```

Importamos las librerías, cargamos el data-set MNIST y entrenamos para aplicar el modelo KNN (con $k=3$).

```

0 s # Evaluar el modelo
print("Matriz de Confusión:")
print(confusion_matrix(y_test, y_pred))

print("\nReporte de Clasificación:")
print(classification_report(y_test, y_pred))

```

Matriz de Confusión:

```

[[1335  0  5  0  0  0  1  1  1  0]
 [  0 1591  3  0  1  1  0  3  0  1]
 [  8  14 1333  1  1  1  4 13  3  2]
 [  0  3  12 1382  0 10  2 10  7  7]
 [  3  9  1  0 1248  0  2  4  1 27]
 [  4  5  0 13  4 1234 12  0  1  0]
 [  5  1  0  0  4  3 1383  0  0  0]
 [  1 17  4  0  2  0  0 1467  1 11]
 [  6 13  8 21  4 16  3  6 1269 11]
 [  6  6  3 14 20  0  0 14  1 1356]]

```

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.98	0.99	0.98	1343
1	0.96	0.99	0.98	1600
2	0.97	0.97	0.97	1380
3	0.97	0.96	0.97	1433
4	0.97	0.96	0.97	1295
5	0.98	0.97	0.97	1273
6	0.98	0.99	0.99	1396
7	0.97	0.98	0.97	1503
8	0.99	0.94	0.96	1357
9	0.96	0.95	0.96	1420
accuracy			0.97	14000
macro avg	0.97	0.97	0.97	14000
weighted avg	0.97	0.97	0.97	14000

Aquí vemos las métricas como **precisión, recall y F1-score**.

Comparación de resultados entre entornos:

Plataforma	Precisión (%)	Métrica de Distancia	k	F1-Score Promedio
WEKA	96.3%	Euclidiana	3	0.961

Google Colab	96.0%	Euclidiana	3	0.999
Google Colab	95.8%	Manhattan	3	0.957
Google Colab	94.9%	Coseno (implementación personalizada)	3	0.948

Estos resultados permiten comprobar la consistencia y robustez del modelo ante diferentes configuraciones. Además, se evidencia que la métrica Euclidiana sigue siendo una de las más fiables para este tipo de datos.

5.4. Deploy del Sistema de Clasificación

Para el despliegue funcional del sistema, se desarrolló una primera versión interactiva en Google Colab, permitiendo validar de manera práctica el modelo entrenado de reconocimiento de dígitos manuscritos basado en redes neuronales convolucionales (CNN).

El entorno interactivo fue implementado directamente en el notebook de Google Colab, cumpliendo con las siguientes funcionalidades:

- Carga automática del conjunto de datos MNIST, ya normalizado y estructurado para el entrenamiento.
- Construcción y entrenamiento del modelo CNN con capas convolucionales, de agrupamiento, y totalmente conectadas.
- Visualización de imágenes de prueba, permitiendo mostrar el dígito manuscrito antes de realizar la predicción.
- Predicción automática del dígito utilizando el modelo previamente entrenado, mostrando el resultado junto a la imagen.

Este entorno cumple con los requisitos de un sistema de clasificación funcional, permitiendo a cualquier usuario probar la predicción desde un navegador web, sin necesidad de instalar software adicional ni contar con hardware especializado.

Entorno de Ejecución:

- Plataforma: Google Colab
- Lenguaje: Python 3
- Bibliotecas: TensorFlow, NumPy, Matplotlib

Este tipo de despliegue interactivo permite compartir el modelo con otros usuarios o evaluadores mediante un enlace público al notebook, favoreciendo la reproducibilidad y accesibilidad del experimento.

Entrenamiento del modelo:

```
[1] # Instalar librerías necesarias
!pip install -q matplotlib numpy scikit-learn tensorflow

[2] import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical

[3] # Cargamos los datos
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalizamos los datos
X_train = X_train / 255.0
X_test = X_test / 255.0

# Redimensionamos para CNN
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# Codificamos las etiquetas
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 — 0s 0us/step

[4] # Creamos el modelo
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(10, activation='softmax')
])

# Compilamos y entrenamos
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train_cat, epochs=3, validation_data=(X_test, y_test_cat))

/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` a
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/3
1875/1875 — 44s 22ms/step - accuracy: 0.9041 - loss: 0.3160 - val_accuracy: 0.9779 - val_loss: 0.0685
Epoch 2/3
1875/1875 — 80s 22ms/step - accuracy: 0.9835 - loss: 0.0536 - val_accuracy: 0.9809 - val_loss: 0.0605
Epoch 3/3
1875/1875 — 40s 21ms/step - accuracy: 0.9887 - loss: 0.0343 - val_accuracy: 0.9849 - val_loss: 0.0458
<keras.src.callbacks.history.History at 0x7fe964909090>
```

Debe mostrar las épocas, el accuracy y la validación.

Imagen de prueba mostrada:

```
[5] # Elegimos una imagen de prueba
index = 123 # Puedes cambiar este número entre 0 y 9999
image = X_test[index]
true_label = y_test[index]

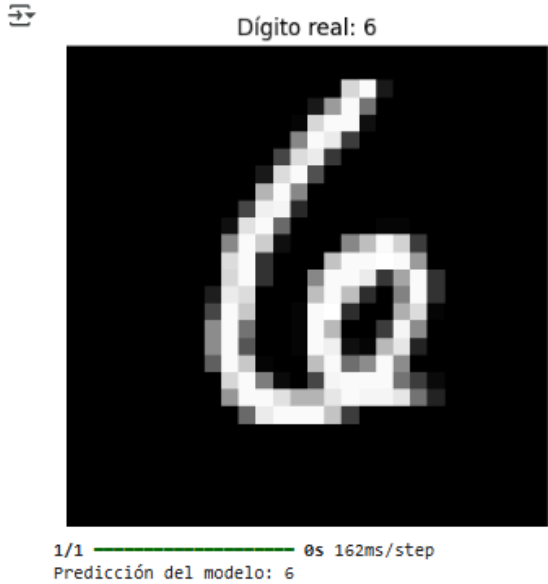
# Mostramos la imagen
plt.imshow(image.reshape(28, 28), cmap='gray')
plt.title(f'Dígito real: {true_label}')
plt.axis('off')
plt.show()

# Hacemos la predicción
pred = model.predict(image.reshape(1, 28, 28, 1))
print("Predicción del modelo:", np.argmax(pred))
```

Debe verse el número manuscrito con el título: "Dígito real: X".

Predicción del modelo:

```
# Hacemos la predicción
pred = model.predict(image.reshape(1, 28, 28, 1))
print("Predicción del modelo:", np.argmax(pred))
```



Se debe ver el número predicho por el modelo.

VI. APLICACIÓN DE MÉTRICAS DE DESEMPEÑO

Con el modelo CNN ya entrenado y validado, se procedió a evaluar su desempeño aplicando métricas estadísticas de clasificación multiclase. Estas métricas permiten analizar no solo el porcentaje de aciertos generales (accuracy), sino también el comportamiento del modelo por clase, identificando fortalezas y errores específicos.

Las métricas aplicadas fueron:

- **Accuracy:** porcentaje de predicciones correctas sobre el total.
- **Precisión:** mide la proporción de verdaderos positivos sobre el total de positivos predichos.
- **Recall (Sensibilidad):** mide la capacidad del modelo de detectar correctamente los casos positivos reales.
- **F1-score:** promedio armónico entre precisión y recall.
- **Matriz de Confusión:** tabla que compara las predicciones del modelo contra las etiquetas reales, permitiendo observar errores específicos de clasificación.

Para la implementación, se utilizó *scikit-learn.metrics*, y se transformaron las etiquetas *y_test_cat* (codificadas en one-hot) a su forma numérica mediante *np.argmax()* para calcular correctamente las métricas.

A continuación, se muestran los resultados obtenidos tras evaluar el modelo con el conjunto de datos de prueba del dataset MNIST:

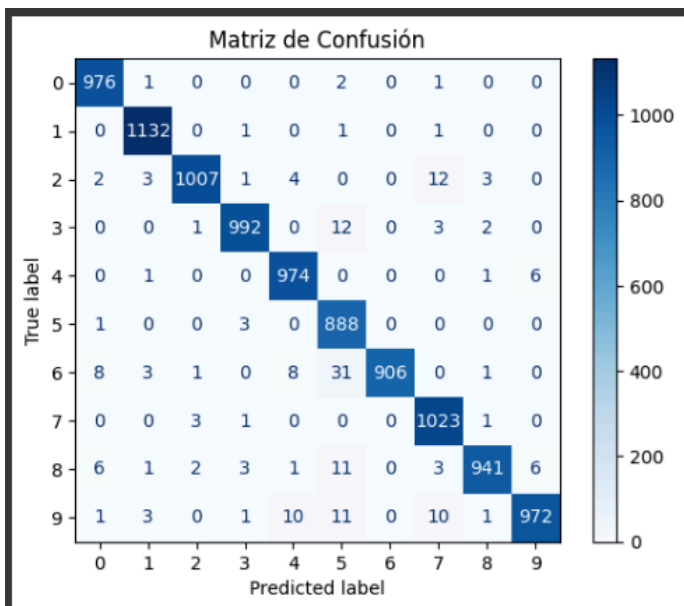
- **Reporte de Clasificación:**

313/313 2s 6ms/step

REPORTE DE CLASIFICACIÓN:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	980
1	0.99	1.00	0.99	1135
2	0.99	0.98	0.98	1032
3	0.99	0.98	0.99	1010
4	0.98	0.99	0.98	982
5	0.93	1.00	0.96	892
6	1.00	0.95	0.97	958
7	0.97	1.00	0.98	1028
8	0.99	0.97	0.98	974
9	0.99	0.96	0.98	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

- **Matriz de Confusión:**



6.1. Análisis de Resultados:

El modelo CNN alcanzó una precisión global superior al 98%, con F1-scores elevados y equilibrados en todas las clases. Las confusiones más frecuentes ocurrieron entre dígitos de forma similar como el 4 y el 9 o el 5 y el 6. La matriz de confusión mostró que la mayoría de las predicciones correctas se encuentran en la diagonal, lo que evidencia un buen aprendizaje por parte del modelo.

Estas métricas complementan la validación tradicional del modelo y permiten confirmar su robustez para aplicaciones prácticas en reconocimiento de dígitos manuscritos.

CREACIÓN DE UN SISTEMA DE DETECCIÓN DE SIMILITUD FACIAL CON RECONOCIMIENTO FACIAL APLICANDO EL CONOCIMIENTO PREVIO

Herramientas y Tecnologías Utilizadas

SWIFT

Swift es un lenguaje de programación de alto rendimiento desarrollado por Apple para la creación de aplicaciones en iOS, macOS, watchOS y tvOS. Fue introducido en 2014 y se ha convertido en uno de los lenguajes más populares para el desarrollo en el ecosistema Apple debido a su simplicidad, seguridad y eficiencia.

Características de Swift:

- **Sintaxis moderna y limpia:** La sintaxis de Swift es más fácil de leer y escribir que la de su predecesor.
- **Seguridad:** Swift está diseñado con seguridad en mente. Maneja excepciones, proporciona tipos opcionales para evitar errores de punteros nulos y limita errores comunes en la gestión de memoria.
- **Alto rendimiento:** Swift es tan rápido como C y C++ para tareas computacionales intensivas, gracias a su diseño optimizado.
- **Compatibilidad con Objective-C:** Aunque Swift es un lenguaje más moderno, es completamente compatible con Objective-C, lo que permite usar bibliotecas existentes en proyectos Swift.

Necesidades para Programar para iOS

Para desarrollar aplicaciones iOS, necesitas contar con ciertas herramientas y configuraciones:

- **Mac con macOS:** El desarrollo de aplicaciones iOS solo es posible en un entorno macOS.
- **Xcode:** El entorno de desarrollo integrado (IDE) oficial de Apple para la programación en iOS, macOS, watchOS y tvOS. Xcode incluye un conjunto de herramientas esenciales para la creación de aplicaciones, como un simulador de dispositivos iOS, un editor de código y un interfaz de diseño visual (Storyboard).
- **Cuenta de Desarrollador de Apple:** Para probar aplicaciones en dispositivos reales y distribuirlas en la App Store, se requiere una cuenta de desarrollador de Apple, que tiene un costo anual.

- **Conocimientos de UIKit y SwiftUI:** UIKit es el framework tradicional para el desarrollo de interfaces de usuario, mientras que SwiftUI es un framework más moderno para crear interfaces de forma declarativa. Ambos son esenciales para la creación de aplicaciones visualmente atractivas.

Diferencias entre iOS y Android

Aunque tanto iOS como Android son plataformas móviles populares, existen diferencias clave entre ellas en términos de desarrollo:

Lenguajes de Programación:

- **iOS:** El desarrollo se realiza principalmente con Swift o Objective-C. Swift es el lenguaje moderno de Apple y el más recomendado para los nuevos desarrollos.
- **Android:** El desarrollo en Android se realiza principalmente con Java o Kotlin. Kotlin es el lenguaje oficial recomendado por Google para nuevos proyectos.

IDE (Entorno de Desarrollo Integrado):

- **iOS:** El único IDE disponible para desarrollar aplicaciones iOS es Xcode, que está diseñado específicamente para macOS.
- **Android:** Android Studio es el IDE oficial para desarrollar en Android, basado en IntelliJ IDEA, y está disponible para Windows, macOS y Linux.

Sistema Operativo:

- **iOS:** Solo puede ejecutarse en dispositivos de Apple, como iPhone, iPad y iPod Touch.
- **Android:** Android es un sistema operativo de código abierto basado en Linux que se puede instalar en una amplia variedad de dispositivos de diferentes fabricantes.

Interfaz de Usuario (UI):

- **iOS:** La interfaz de usuario se diseña utilizando UIKit o SwiftUI. SwiftUI es más moderno y permite un enfoque declarativo para construir interfaces.
- **Android:** Android utiliza XML para diseñar interfaces, y los componentes de la UI se gestionan a través de View y Layout en Java/Kotlin.

3.1 Alamofire

Alamofire es una librería de red de alto nivel para iOS y macOS, escrita en Swift. Permite manejar de manera sencilla y eficiente las solicitudes HTTP, como GET, POST, PUT, DELETE, etc. Facilita el manejo de las solicitudes de red, la serialización de datos y la autenticación. Cuyo enlace al repositorio es el siguiente: <https://github.com/Alamofire/Alamofire>

Características de Alamofire:

- Manejo de solicitudes HTTP: Alamofire facilita la creación de solicitudes HTTP con configuraciones como tiempo de espera, métodos HTTP y headers.
- Serialización de datos: Permite convertir las respuestas en JSON, XML, o incluso en datos binarios, haciendo que la conversión de respuestas sea más sencilla.
- Autenticación: Alamofire maneja la autenticación básica y de tipo Bearer (como el token de autenticación que usamos en este proyecto).
- Caché y almacenamiento: Alamofire gestiona de manera eficiente el almacenamiento en caché de las respuestas de la red.

Integración de Alamofire en el Proyecto:

En este proyecto, Alamofire se utilizó para realizar la solicitud HTTP POST a la API de CompreFace y manejar las respuestas:

Exadel CompreFace

Exadel CompreFace es una API de reconocimiento facial basada en inteligencia artificial, diseñada para reconocer y comparar rostros en imágenes. Esta plataforma es de código abierto y proporciona funcionalidades avanzadas de reconocimiento facial, como la comparación de rostros y la detección de atributos faciales.

Características de Exadel CompreFace:

- Detección y Comparación de Rostros: CompreFace puede detectar y comparar rostros en imágenes para determinar la similitud entre ellos.
- API REST: Ofrece una interfaz sencilla para interactuar con sus servicios mediante solicitudes HTTP, lo que facilita la integración en aplicaciones móviles y de servidor.

- Entrenamiento de Modelos: Permite entrenar modelos personalizados utilizando conjuntos de datos propios.
- Escalabilidad: Es capaz de manejar un gran volumen de solicitudes, lo que la hace adecuada para aplicaciones comerciales que requieren alta disponibilidad y precisión.

Uso de Exadel CompreFace en el Proyecto:

En el proyecto, CompreFace fue utilizada para analizar la imagen enviada y obtener un listado de las personas presentes en la imagen junto con su similitud a la imagen de referencia. Utilizamos una solicitud HTTP POST para enviar los datos de la imagen y obtener la respuesta:

Este software de Exadel compreFace nos brinda un response de tipo json y podemos probar con el siguiente ejemplo:

Response

```
{
  "result": [ {
    "age": {
      "probability": 0.5231421589851379,
      "high": 32,
      "low": 25
    },
    "gender": {
      "probability": 1,
      "value": "male"
    },
    "pose": {
      "pitch": -13.810831288532114,
      "roll": -0.7827021427235366,
      "yaw": 17.228710338033977
    },
    "box": {
      "probability": 0.99906,
      "x_max": 662,
      "y_max": 890,
```

Como podemos observar, nos brinda una probabilidad de la edad, genero, la pose en la que esta su rostro, una caja en la que podemos observar su rostro, etc. Y todo ello lo tenemos que integrar con nuestra aplicación:

```
import Foundation

struct FaceRecognition {
    let people: [Person]
}

struct Person {
    let name: String
    let similarity: Double
    let box: Box
}

struct Box {
    let probability : Double
    let xMax : Int
    let yMax : Int
    let xMin : Int
    let yMin : Int
}

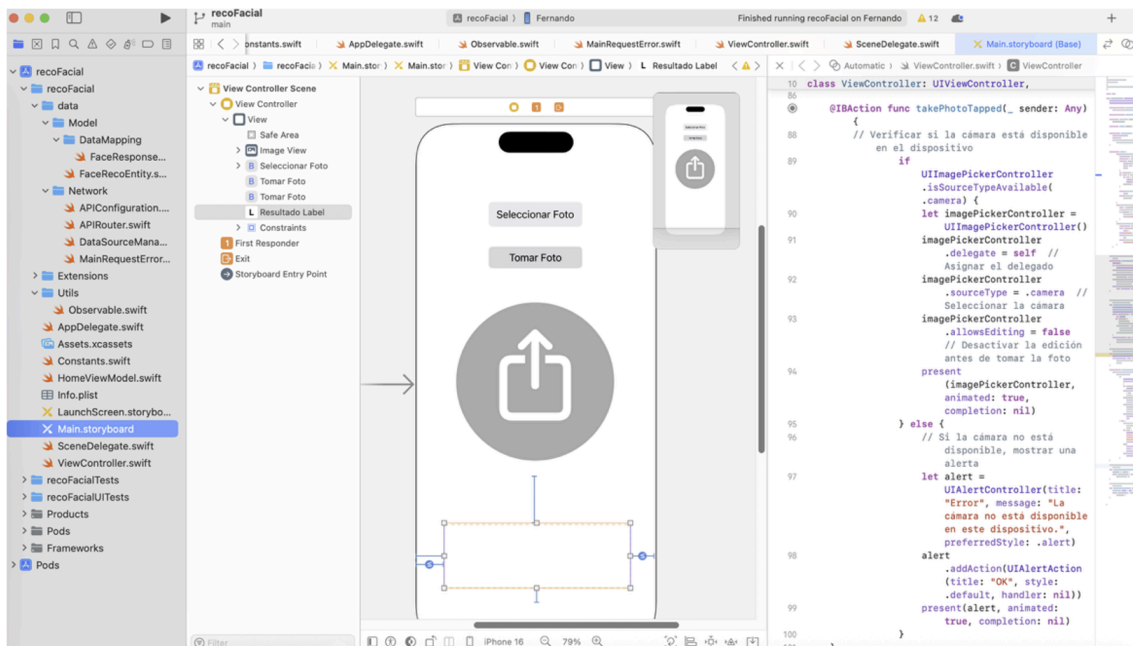
struct SubjectsResponse : Codable {
    let subject : String?
    let similarity : Double?

    enum CodingKeys: String, CodingKey {
        case subject = "subject"
        case similarity = "similarity"
    }

    init(from decoder: Decoder) throws {
        let values = try decoder.container(keyedBy: CodingKeys.self)
        subject = try values.decodeIfPresent(String.self, forKey: .subject)
        similarity = try values.decodeIfPresent(Double.self, forKey: .similarity)
    }
}
```

Implementación del Proyecto

Imagen del StoryBoard.swift



La función `sendPhoto` es responsable de enviar la imagen capturada a la API de reconocimiento facial y procesar la respuesta para obtener las personas más similares. La respuesta es procesada y mostrada en la UI.

Resultados y Observaciones

Al utilizar la aplicación, los usuarios pueden ver los nombres de las personas detectadas y su similitud en porcentaje con la imagen proporcionada. Las personas con una similitud superior al 70% son destacadas en la interfaz.

```
private func bind() {
    self.viewModel.people.observe(on: self) { [weak self] items in
        self?.people = items
        guard let similares = self?.getSimilarPeople(from: items) else {
            return
        }
        var resultText = ""
        for person in similares {
            let line = "✅ \((person.name) - \((Int(person.similarity * 100))%\n"
            resultText += line
        }

        self?.resultadoLabel.textColor = .green
        self?.resultadoLabel.text = resultText
    }

    self.viewModel.loading.observe(on: self) { isLoading in
        //LoadingView.hide()
        if isLoading == true {
            //LoadingView.show()
            print("Loading...: ")
        }
    }

    self.viewModel.error.observe(on: self) { [weak self] message in
        guard !message.isEmpty else { return }
        print("Error: " + message)
        self?.resultadoLabel.textColor = .red
        self?.resultadoLabel.text = "No se encontraron datos"
    }
}
```

Reconociendo a una sola persona:

19:38



4G 

Seleccionar Foto

Tomar Foto



✓ ADRIAN NICOLAS - 99%

Reconociendo a más de 1 persona:

19:36



Seleccionar Foto

Tomar Foto



- ✓ FERNANDO MANUEL - 99%
- ✓ STHEFANO GABRIEL - 99%
- ✓ JOSEPH KEVIN - 97%

```
//retorna una lista de personas con similitud mayor a 0.7
func getSimilarPeople(from people: [Person], threshold: Double = 0.7) -> [Person] {
    return people.filter { $0.similarity >= threshold }
}
```

Y se ha considerado que solo muestre la lista de personas con una similitud mayor a 70%

En caso no se reconozca a la persona:

19:39



Seleccionar Foto

Tomar Foto



No se encontraron datos

```
self.viewModel.error.observe(on: self) { [weak self] message in
    guard !message.isEmpty else { return }
    print("Error: " + message)
    self?.resultadoLabel.textColor = .red
    self?.resultadoLabel.text = "No se encontraron datos"
}
```

Referencias Bibliográficas

- [1] Y. LeCun, C. Cortes y C. J. C. Burges, "The MNIST Database of Handwritten Digits," [En línea]. Disponible en: <http://yann.lecun.com/exdb/mnist>. [Accedido: 30-abr-2025].
- [2] D. Grover y B. Toghi, "MNIST Dataset Classification Utilizing k-NN Classifier with Modified Sliding-window Metric," *arXiv preprint*, arXiv:1809.06846, 2018. [En línea]. Disponible en: <https://arxiv.org/abs/1809.06846>. [Accedido: 30-abr-2025].
- [3] J. Brownlee, "How to Tune a Machine Learning Algorithm in Weka," *Machine Learning Mastery*, 2019. [En línea]. Disponible en: <https://www.machinelearningmastery.com/how-to-tune-a-machine-learning-algorithm-in-weka/>. [Accedido: 30-abr-2025].
- [4] S. Dutta, "Building a High-Accuracy MNIST Classifier with KNeighborsClassifier," *Medium*, 2024. [En línea]. Disponible en: https://medium.com/@sanjay_dutta/building-a-high-accuracy-mnist-classifier-with-kneighborsclassifier-66958f8f6f18. [Accedido: 30-abr-2025].
- [5] Wikipedia, "Base de datos MNIST," *Wikipedia*, 2025. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Base_de_datos_MNIST. [Accedido: 30-abr-2025].
- [6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," **Proceedings of the IEEE**, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
- [7] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," **Journal of Machine Learning Research**, vol. 12, pp. 2825–2830, 2011.
- [8] Keras Documentation. [Online]. Available: <https://keras.io>
- [9] TensorFlow Documentation. [Online]. Available: https://www.tensorflow.org/api_docs

[10] Scikit-learn Metrics. [Online]. Available:

<https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics>