import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import streamlit as st

from sklearn.model\_selection import train\_test\_split

def two\_split(X, y, holdout\_frac=0.2, test\_size=0.2, random\_state=42):

  X\_hold, X\_rest, y\_hold, y\_rest = train\_test\_split(X, y, test\_size=1-holdout\_frac, random\_state=random\_state)

  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_rest, y\_rest, test\_size=test\_size, random\_state=random\_state)

  return X\_hold, X\_train, X\_test, y\_hold, y\_train, y\_test

def report\_stats(y\_true, y\_pred, prefix=""):

  mae=mean\_absolute\_error(y\_true, y\_pred)

  mse=mean\_squared\_error(y\_true, y\_pred)

  rsme=mse\*\*0.5

  r2=r2\_score(y\_true,y\_pred)

  print(f"{prefix} MAE: {mae:.2f}")

  print(f"{prefix} MSE: {mse:.2f}")

  print(f"{prefix} RSME: {rsme:.2f}")

  print(f"{prefix} R2: {r2:.2f}")

  return{"mae": mae,"mse": mse, "rsme": rsme, "r2": r2 }

(x\_train, y\_train), (x\_test, y\_test) = datasets.mnist.load\_data()

x\_train, x\_test =x\_train/ 255.0, x\_test/ 255.0

x\_train=x\_train.reshape((x\_train.shape[0], 28, 28, 1))

x\_test=x\_test.reshape((x\_test.shape[0], 28, 28, 1))

X=tf.concat([x\_train, x\_test], axis=0)

y=tf.concat([y\_train, y\_test], axis=0)

X\_hold, X\_train, X\_test, y\_hold, y\_train, y\_test = two\_split(X.numpy(), y.numpy(), holdout\_frac=0.2, test\_size=0.2, random\_state=42)

class\_names = ["0","1","2","3","4","5","6","7","8","9"]

plt.figure(figsize=(10,10))

for i in range(20):

  plt.subplot(5,5, i+1)

  plt.xticks([])

  plt.yticks([])

  plt.grid(False)

  plt.imshow(X\_train[i])

  plt.xlabel(class\_names[y\_train[i]])

plt.show()

model= models.Sequential()

model.add(layers.Conv2D(32, (3,3), activation='relu', input\_shape=(28, 28, 1)))

model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Conv2D(64, (3,3), activation="relu"))

model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Conv2D(64, (3,3), activation="relu"))

model.summary()

model.add(layers.Flatten())

model.add(layers.Dense(64, activation="relu"))

model.add(layers.Dense(10))

model.summary()

model.compile(optimizer="adam", loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits= True), metrics=["accuracy"])

history=model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

holdout\_loss, holdout\_acc = model.evaluate(X\_hold, y\_hold, verbose=2)

print(f"Final Holdout Loss: {holdout\_loss:.4f} | Accuracy: {holdout\_acc:.4f}")

**Model: "sequential\_1"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩

│ conv2d\_3 (Conv2D) │ (None, 26, 26, 32) │ 320 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d\_2 (MaxPooling2D) │ (None, 13, 13, 32) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d\_4 (Conv2D) │ (None, 11, 11, 64) │ 18,496 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d\_3 (MaxPooling2D) │ (None, 5, 5, 64) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d\_5 (Conv2D) │ (None, 3, 3, 64) │ 36,928 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

**Total params:** 55,744 (217.75 KB)

**Trainable params:** 55,744 (217.75 KB)

**Non-trainable params:** 0 (0.00 B)

**Model: "sequential\_1"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩

│ conv2d\_3 (Conv2D) │ (None, 26, 26, 32) │ 320 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d\_2 (MaxPooling2D) │ (None, 13, 13, 32) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d\_4 (Conv2D) │ (None, 11, 11, 64) │ 18,496 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d\_3 (MaxPooling2D) │ (None, 5, 5, 64) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d\_5 (Conv2D) │ (None, 3, 3, 64) │ 36,928 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ flatten\_1 (Flatten) │ (None, 576) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_2 (Dense) │ (None, 64) │ 36,928 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_3 (Dense) │ (None, 10) │ 650 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

**Total params:** 93,322 (364.54 KB)

**Trainable params:** 93,322 (364.54 KB)

**Non-trainable params:** 0 (0.00 B)

Epoch 1/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **51s** 35ms/step - accuracy: 0.8763 - loss: 0.4034 - val\_accuracy: 0.9767 - val\_loss: 0.0752

Epoch 2/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **49s** 35ms/step - accuracy: 0.9828 - loss: 0.0548 - val\_accuracy: 0.9825 - val\_loss: 0.0618

Epoch 3/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **47s** 34ms/step - accuracy: 0.9878 - loss: 0.0361 - val\_accuracy: 0.9869 - val\_loss: 0.0433

Epoch 4/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **81s** 33ms/step - accuracy: 0.9921 - loss: 0.0258 - val\_accuracy: 0.9835 - val\_loss: 0.0608

Epoch 5/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **48s** 34ms/step - accuracy: 0.9925 - loss: 0.0211 - val\_accuracy: 0.9861 - val\_loss: 0.0519

Epoch 6/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **48s** 35ms/step - accuracy: 0.9948 - loss: 0.0154 - val\_accuracy: 0.9886 - val\_loss: 0.0520

Epoch 7/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **48s** 34ms/step - accuracy: 0.9967 - loss: 0.0110 - val\_accuracy: 0.9889 - val\_loss: 0.0450

Epoch 8/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **48s** 34ms/step - accuracy: 0.9974 - loss: 0.0081 - val\_accuracy: 0.9877 - val\_loss: 0.0512

Epoch 9/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **80s** 33ms/step - accuracy: 0.9966 - loss: 0.0092 - val\_accuracy: 0.9894 - val\_loss: 0.0525

Epoch 10/10

**1400/1400** ━━━━━━━━━━━━━━━━━━━━ **47s** 34ms/step - accuracy: 0.9972 - loss: 0.0086 - val\_accuracy: 0.9906 - val\_loss: 0.0446

438/438 - 4s - 9ms/step - accuracy: 0.9899 - loss: 0.0420

Final Holdout Loss: 0.0420 | Accuracy: 0.9899

**Report on the undertaking and findings of the CNN Model**

This report summarizes the structure, training, and performance of a Convolutional Neural Network (CNN) model (sequential\_1) used for image classification of the MNIST dataset

The first convolutional layer, a Conv2D type, extracts 32 feature maps using small filters (probably 3×3). “None” means batch size is flexible.

The second convolutional layer hash 64 filters and learns more complex features.

The third convolutional layer performs deeper feature extraction with 64 filters.

There are maxpooling steps between each convolution to downsample features by taking the max value in each 2×2 window, reducing size and computation. This prevents overfitting.

After convolutional steps, the images are flattened from 3D to 1D vectors for dense layers.

Two densing layers follow:

1. dense\_2 which is a fully connected layer with 64 neurons to combine extracted features.
2. dense\_3 which is an output layer for classification into 10 categories (digits 0–9).

**Parameter Breakdown**

Conv2D parameters formula:

(kernel height×kernel width×input channels+1)×number of filters

The “+1” is for the bias term per filter.

Example: For conv2d\_3, assuming input = 1 channel (grayscale): (3×3×1+1)×32=320

The numbers match perfectly with the summary

**Model Training Results**

There were 10 training cycles (epochs) where the model trained efficiently without overfitting

Training accuracy: 0.9972. Accuracy on training data. Extremely high (99.7%), showing excellent learning.

Validation accuracy: 0.9906. Accuracy on unseen validation data (99.1%). This shows that it generalizes well.

Holdout accuracy: 0.9899. This is data from the dataset that was kept away from the model to be used for final evaluation after training and validation. High percentage (98.9%) shows that the model can work well with totally new data.

Loss value: Training loss dropped from 0.4034 to 0.0086. Smooth convergence, no major divergence between train and validation losses (no overfitting). indication seamless learning by the model with correct and realistic output.

CNN successfully learned key visual features; strong model for classification tasks like MNIST.