Upload the Dataset

from google.colab import files

uploaded = files.upload()

Load the Dataset

from tensorflow.keras.datasets

import mnist # Load dataset

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

Data Exploration print("Training data shape:", x\_train.shape)

print("Test data shape:", x\_test.shape)

print("Sample labels:", y\_train[:10])

Check for Missing Values and Duplicates

# MNIST dataset is clean; still, we check

import numpy as np print("Missing values in train data:", np.isnan(x\_train).sum()) print("Duplicates in train labels:", len(y\_train) - len(np.unique(y\_train)))

Visualize a Few Features

import matplotlib.pyplot as plt

for i in range(9):

plt.subplot(3, 3, i+1)

plt.imshow(x\_train[i], cmap='gray')

plt.title(f"Label: {y\_train[i]}")

plt.axis('off')

plt.tight\_layout()

plt.show()

Identify Target and Features

# Features (input): pixel values of 28x28 images

print("Feature shape (x\_train):", x\_train.shape)

# Target (output): digit labels (0-9)

print("Target shape (y\_train):", y\_train.shape)

print("Unique labels (classes):", np.unique(y\_train))

Convert Categorical Columns to Numerical

from tensorflow.keras.utils import to\_categorical

# Convert categorical integer labels to one-hot encoded vectors

y\_train\_encoded = to\_categorical(y\_train, num\_classes=10)

y\_test\_encoded = to\_categorical(y\_test, num\_classes=10)

# Confirm shape

print("One-hot encoded y\_train shape:", y\_train\_encoded.shape)

print("Example (label 5):", y\_train\_encoded[5])

One-Hot Encoding

from tensorflow.keras.utils import to\_categorical

y\_train\_encoded = to\_categorical(y\_train)

y\_test\_encoded = to\_categorical(y\_test)

Feature Scaling

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

Train-Test Split

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

# First, reshape and normalize image data

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

x\_train\_cnn = x\_train.reshape(-1, 28, 28, 1)

x\_test\_cnn = x\_test.reshape(-1, 28, 28, 1)

# One-hot encode labels

from tensorflow.keras.utils import to\_categorical

y\_train\_encoded = to\_categorical(y\_train, num\_classes=10) y\_test\_encoded = to\_categorical(y\_test, num\_classes=10)

# Split training data into new training and validation sets (e.g., 90% train, 10% validation) x\_train\_final, x\_val, y\_train\_final, y\_val = train\_test\_split( x\_train\_cnn, y\_train\_encoded, test\_size=0.1, random\_state=42 )

# Print the shapes print("Train data shape:", x\_train\_final.shape)

print("Validation data shape:", x\_val.shape)

print("Test data shape:", x\_test\_cnn.shape)

Model Building

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Reshape for CNN

x\_train\_cnn = x\_train.reshape(-1, 28, 28, 1)

x\_test\_cnn = x\_test.reshape(-1, 28, 28, 1)

model = Sequential([

Conv2D(32, (3,3), activation='relu', input\_shape=(28,28,1)),

MaxPooling2D(2,2),

Flatten(), Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) model.summary()

Evaluation

history = model.fit(x\_train\_cnn, y\_train\_encoded, epochs=5, validation\_split=0.1)

# Evaluate

test\_loss, test\_acc = model.evaluate(x\_test\_cnn, y\_test\_encoded)

print(f"\nTest Accuracy: {test\_acc:.2f}")

Make Predictions from New Input

predictions = model.predict(x\_test\_cnn)

predicted\_labels = np.argmax(predictions, axis=1)

Convert to DataFrame and Encode

import pandas as pd df\_preds = pd.DataFrame({'Actual': y\_test, 'Predicted': predicted\_labels})

df\_preds.head(10)

Predict the Final Grade (i.e., Digit)

# Show one prediction

plt.imshow(x\_test[0], cmap='gray')

View recommended plots  plt.title(f"Model Prediction: {predicted\_labels[0]}")

plt.axis('off')

plt.show()

Deployment – Building an Interactive App

!pip install gradio

import gradio as gr

Create a Prediction Function

def predict\_digit(image):

import numpy as np

img = image.reshape(1, 28, 28, 1)

img = img / 255.0

pred = model.predict(img)

return np.argmax(pred)

Create the Gradio Interface

# Step 1: Install Gradio (if not already installed)

!pip install gradio –quiet

# Step 2: Import libraries

import gradio as gr

import numpy as np

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.utils import to\_categorical

from PIL import Image

# Step 3: Load and preprocess MNIST data

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

x\_train = x\_train.reshape(-1, 28, 28, 1) / 255.0

y\_train\_cat = to\_categorical(y\_train)

# Step 4: Build and train model

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) model.fit(x\_train, y\_train\_cat, epochs=1)

# Step 5: Prediction function

def predict\_digit(img):

img = img.convert('L').resize((28, 28))

# Convert to grayscale and resize

img\_array = 255 - np.array(img)

# Invert image (white bg to black bg)

img\_array = img\_array / 255.0

# Normalize img\_array = img\_array.reshape(1, 28, 28, 1)

# Reshape for model prediction = model.predict(img\_array)

return f"Predicted Digit: {np.argmax(prediction)}"

# Step 6: Gradio interface (NO deprecated args)

gr.Interface(

fn=predict\_digit,

inputs=gr.Image(type="pil", image\_mode="L"),

outputs="text",

title="Handwritten Digit Recognition",

description="Draw a digit (0-9) and get the predicted result!"

).launch(share=True)