**HANDWRITTEN DIGIT RECOGNIZATION WITH DEEP LEARNING FOR SMARTER AI APPLICATION**

**PROJECT CODE AND OUTPUT:**

#IMPORING LIBRARIES

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.metrics import classification\_report, confusion\_matrix

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.utils import to\_categorical

import streamlit as st

#LOADING DATASET

train\_df = pd.read\_csv('mnist\_train.csv')

test\_df = pd.read\_csv('mnist\_test.csv')

#SEPERATE FEATURES AND LABELS

x\_train = train\_df.drop('label', axis=1).values

y\_train = train\_df['label'].values

x\_test = test\_df.drop('label', axis=1).values

y\_test = test\_df['label'].values

#NORMALIZE PIXEL VALUES

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

#RESHAPE FOR CNN INPUT

# Reshape for CNN input

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

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

#ONE-HOT ENCODE LABELS

y\_train\_cat = to\_categorical(y\_train, 10)

y\_test\_cat = to\_categorical(y\_test, 10)

#PLOTTING SAMPLE DIGITS

plt.figure(figsize=(8,6))

for i in range(9):

    plt.subplot(3, 3, i+1)

    plt.imshow(x\_train[i].reshape(28,28), cmap='gray')

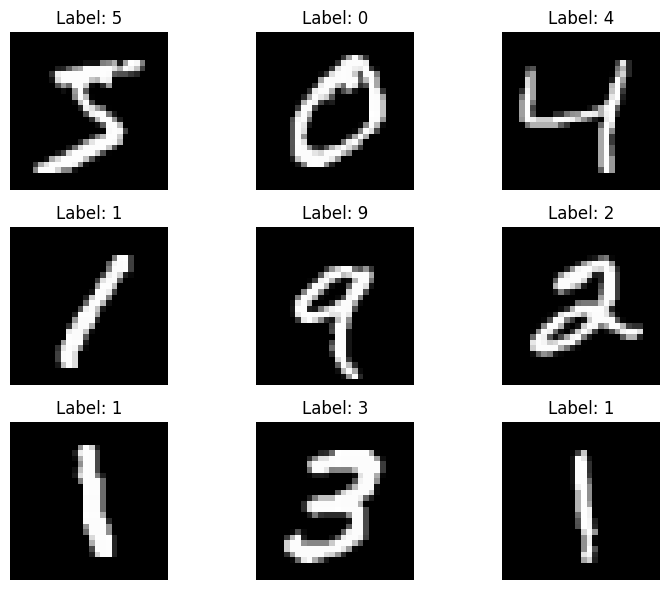
    plt.title(f"Label: {y\_train[i]}")

    plt.axis('off')

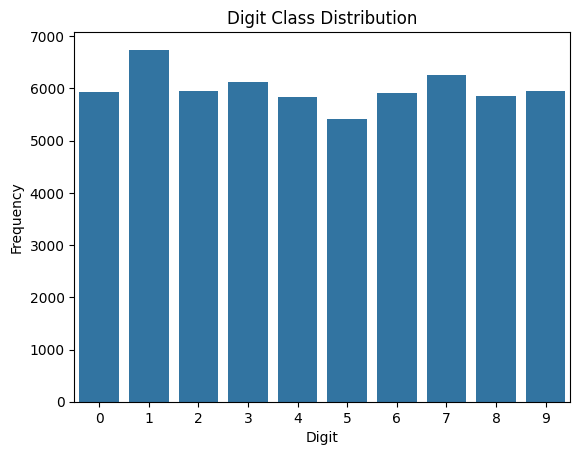
plt.tight\_layout()

plt.show()

#OUTPUT (PLOTTING SAMPLE DIGITS)



#OUTPUT (DIGIT DISTRIBUTION)



#MODEL BUILDING

model = Sequential([

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

    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),

    MaxPooling2D(2,2),

    Flatten(),

    Dense(128, activation='relu'),

    Dropout(0.5),

    Dense(10, activation='softmax')

])

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

model.summary()

for layer in model.layers:

    print(f"{layer.name}: {layer.get\_weights()[0].shape if layer.get\_weights() else 'No weights'}")

#OUTPUT (MODEL BUILDING)

**Model: "sequential\_1"**

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

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

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

│ conv2d\_2 (Conv2D) │ (None, 26, 26, 32) │ 320 │

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

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

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

│ conv2d\_3 (Conv2D) │ (None, 11, 11, 64) │ 18,496 │

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

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

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

│ flatten\_1 (Flatten) │ (None, 1600) │ 0 │

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

│ dense\_2 (Dense) │ (None, 128) │ 204,928 │

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

│ dropout\_1 (Dropout) │ (None, 128) │ 0 │

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

│ dense\_3 (Dense) │ (None, 10) │ 1,290 │

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

**Total params:** 225,034 (879.04 KB)

**Trainable params:** 225,034 (879.04 KB)

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

conv2d\_2: (3, 3, 1, 32)

max\_pooling2d\_2: No weights

conv2d\_3: (3, 3, 32, 64)

max\_pooling2d\_3: No weights

flatten\_1: No weights

dense\_2: (1600, 128)

dropout\_1: No weights

dense\_3: (128, 10)

#MODEL TRAINING AND EVALUATION

# Train model

history = model.fit(x\_train, y\_train\_cat, epochs=10, batch\_size=128, validation\_split=0.1)

# Evaluate on test data

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

print(f"Test Accuracy: {test\_acc \* 100:.2f}%")

#OUTPUT (MODEL BUILDING AND EVALUAION)

Epoch 1/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **46s** 103ms/step - accuracy: 0.7973 - loss: 0.6355 - val\_accuracy: 0.9823 - val\_loss: 0.0597

Epoch 2/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **87s** 117ms/step - accuracy: 0.9678 - loss: 0.1078 - val\_accuracy: 0.9863 - val\_loss: 0.0479

Epoch 3/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **72s** 93ms/step - accuracy: 0.9772 - loss: 0.0749 - val\_accuracy: 0.9897 - val\_loss: 0.0347

Epoch 4/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **42s** 95ms/step - accuracy: 0.9812 - loss: 0.0628 - val\_accuracy: 0.9900 - val\_loss: 0.0313

Epoch 5/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **40s** 93ms/step - accuracy: 0.9845 - loss: 0.0476 - val\_accuracy: 0.9890 - val\_loss: 0.0369

Epoch 6/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **41s** 94ms/step - accuracy: 0.9867 - loss: 0.0438 - val\_accuracy: 0.9900 - val\_loss: 0.0334

Epoch 7/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **41s** 94ms/step - accuracy: 0.9892 - loss: 0.0368 - val\_accuracy: 0.9908 - val\_loss: 0.0279

Epoch 8/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **41s** 95ms/step - accuracy: 0.9902 - loss: 0.0332 - val\_accuracy: 0.9907 - val\_loss: 0.0317

Epoch 9/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **42s** 96ms/step - accuracy: 0.9899 - loss: 0.0313 - val\_accuracy: 0.9917 - val\_loss: 0.0296

Epoch 10/10

**422/422** ━━━━━━━━━━━━━━━━━━━━ **41s** 96ms/step - accuracy: 0.9921 - loss: 0.0255 - val\_accuracy: 0.9922 - val\_loss: 0.0306

**313/313** ━━━━━━━━━━━━━━━━━━━━ **3s** 8ms/step - accuracy: 0.9881 - loss: 0.0370

#ACCURACY OF MODEL (TESTING)

Test Accuracy: 99.09%

#RESULT VIZUALIZATION

->ACCURACY AND LOSS CURVE

Code:

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)

plt.plot(history.history['accuracy'], label='Train')

plt.plot(history.history['val\_accuracy'], label='Val')

plt.title('Model Accuracy')

plt.legend()

plt.subplot(1,2,2)

plt.plot(history.history['loss'], label='Train')

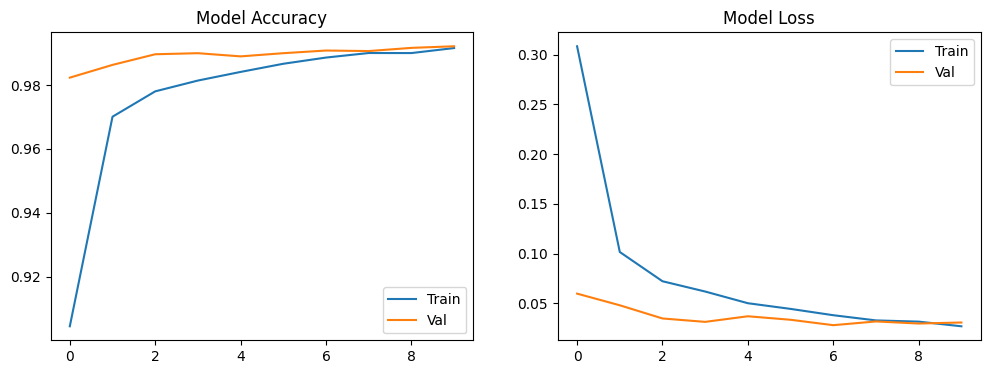
plt.plot(history.history['val\_loss'], label='Val')

plt.title('Model Loss')

plt.legend()

plt.show()

Output:



->CONFUSION MATRIX AND CLASSIFICATION REPORT

Code:

y\_pred = model.predict(x\_test).argmax(axis=1)

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

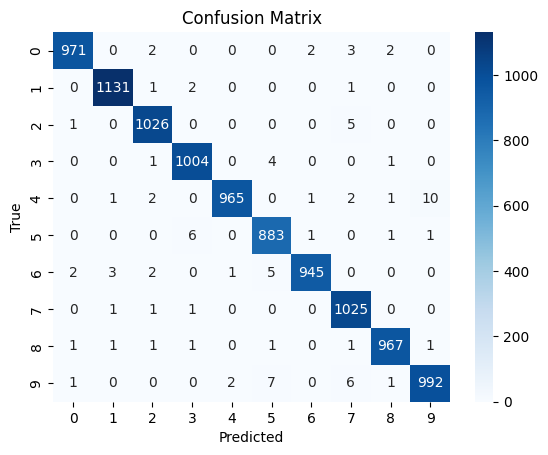
plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("True")

plt.show()

Output:



Classification Report:

precision recall f1-score support

0 0.99 0.99 0.99 980

1 0.99 1.00 1.00 1135

2 0.99 0.99 0.99 1032

3 0.99 0.99 0.99 1010

4 1.00 0.98 0.99 982

5 0.98 0.99 0.99 892

6 1.00 0.99 0.99 958

7 0.98 1.00 0.99 1028

8 0.99 0.99 0.99 974

9 0.99 0.98 0.99 1009

accuracy 0.99 10000

macro avg 0.99 0.99 0.99 10000

weighted avg 0.99 0.99 0.99 10000

#TO SAVE MODEL(AS KERAS )

model.save('digit\_model.keras', include\_optimizer=False)