1.Upload the Dataset

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist

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

# Normalize the input data to [0, 1] range
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape data to fit model input (if using CNNs)
x_train = x_train.reshape((-1, 28, 28, 1))
x_test = x_test.reshape((-1, 28, 28, 1))

# Print dataset shape
print(f"Training data shape: {x_train.shape}, Labels: {y_train.shape}")
print(f"Test data shape: {x_test.shape}, Labels: {y_test.shape}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Training data shape: (60000, 28, 28, 1), Labels: (60000,)
Test data shape: (10000, 28, 28, 1), Labels: (10000,)
```

2.Load the Dataset

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist

# Load the MNIST dataset (downloads it if not already present)
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# Reshape for CNN input (batch_size, height, width, channels)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

print("Dataset loaded successfully.")
print("Training data shape:", x_train.shape)
print("Test data shape:", x_test.shape)
```

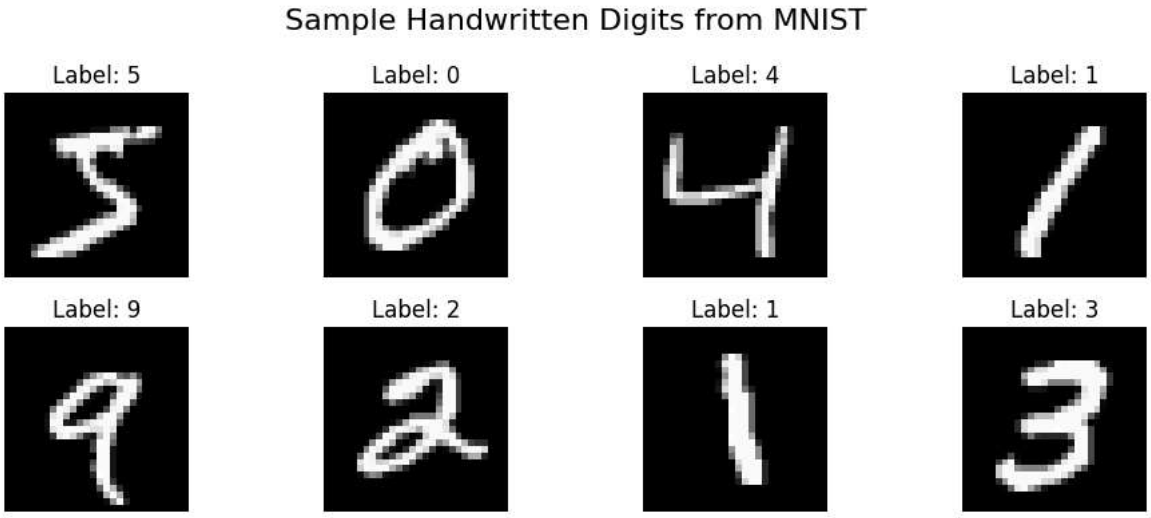
```
Dataset loaded successfully.
Training data shape: (60000, 28, 28, 1)
Test data shape: (10000, 28, 28, 1)
```

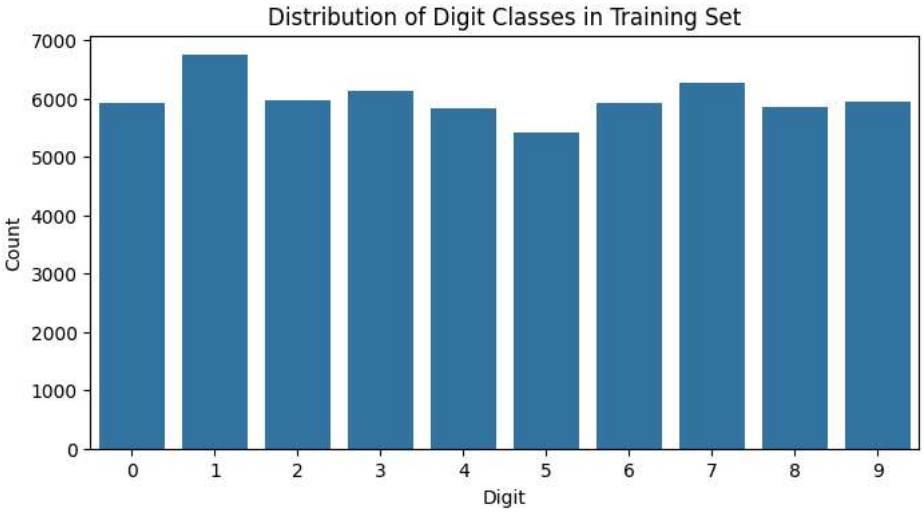
3.Data Exploration

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.datasets import mnist

# Load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Display dataset shapes
print("Training data shape:", x_train.shape)
print("Test data shape:", x_test.shape)
print("Training labels shape:", y_train.shape)

# Check for any missing values
print("Missing values in training data:", np.isnan(x_train).sum())
print("Missing values in test data:", np.isnan(x_test).sum())

# Display the first few images
```

```
plt.figure(figsize=(10, 4))
for i in range(8):
    plt.subplot(2, 4, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.suptitle("Sample Handwritten Digits from MNIST", fontsize=16)
plt.tight_layout()
plt.show()

# Check class distribution
plt.figure(figsize=(8, 4))
sns.countplot(x=y_train)
plt.title("Distribution of Digit Classes in Training Set")
plt.xlabel("Digit")
plt.ylabel("Count")
plt.show()

# Display pixel value stats
print("Pixel value range in training set: min =", x_train.min(), ", max =", x_train.max())
print("Mean pixel value:", np.mean(x_train))
print("Standard deviation of pixel values:", np.std(x_train))
```

```
Training data shape: (60000, 28, 28)
Test data shape: (10000, 28, 28)
Training labels shape: (60000,)
Missing values in training data: 0
Missing values in test data: 0
```






```
Pixel value range in training set: min = 0 , max = 255
Mean pixel value: 33.318421449829934
Standard deviation of pixel values: 78.5674809832970
```

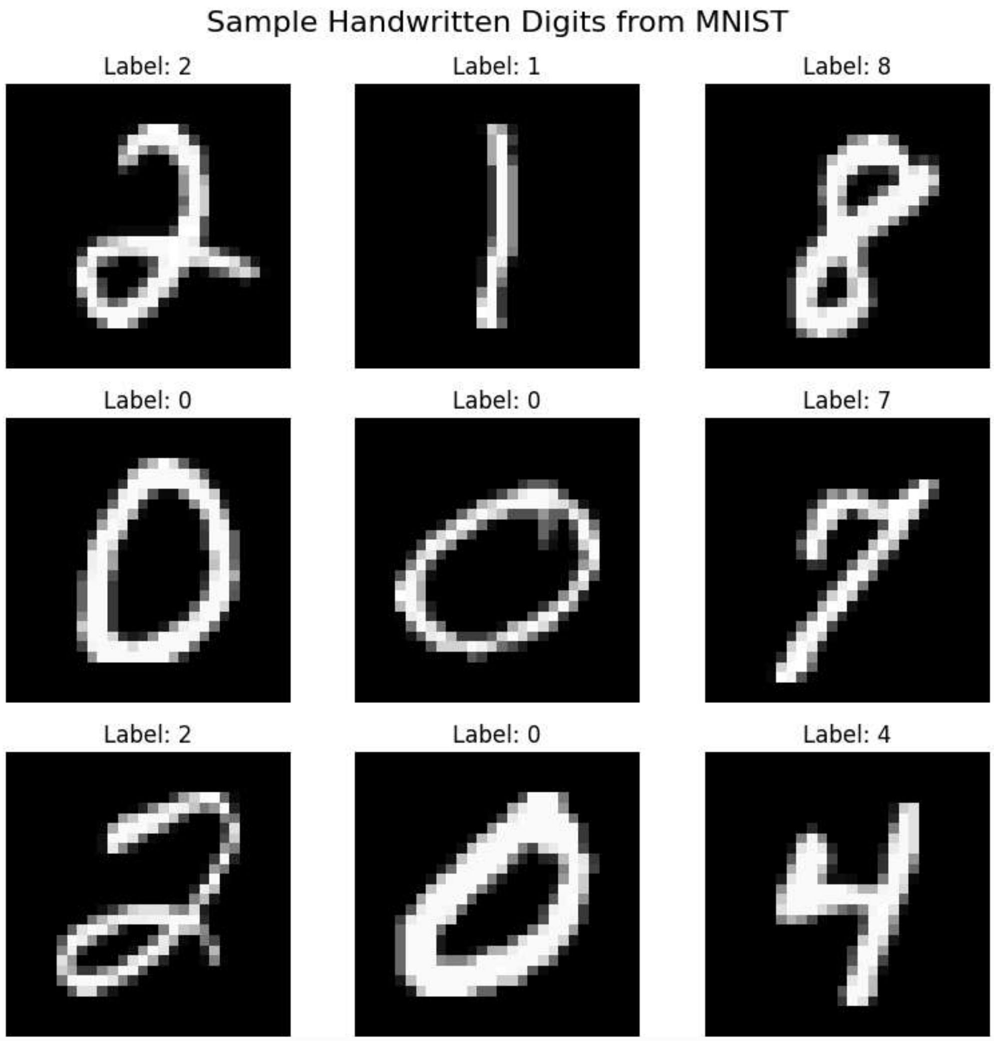
4.Check for Missing Values and Duplicates

```
import numpy as np
from tensorflow.keras.datasets import mnist

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

# 1. Check for missing values (NaNs)
print("Checking for missing values...")
print("Missing values in x_train:", np.isnan(x_train).sum())
print("Missing values in x_test:", np.isnan(x_test).sum())
print("Missing values in y_train:", np.isnan(y_train).sum())
print("Missing values in y_test:", np.isnan(y_test).sum())

# 2. Check for duplicate images in training set
print("\nChecking for duplicate images in x_train...")
# Flatten the images to 1D vectors for comparison
x_train_flat = x_train.reshape((x_train.shape[0], -1))
# Stack image and label pairs to identify duplicates
train_data_combined = np.hstack((x_train_flat, y_train.reshape(-1, 1)))
# Use np.unique to find duplicates
_, unique_indices = np.unique(train_data_combined, axis=0, return_index=True)
num_duplicates = x_train.shape[0] - unique_indices.shape[0]
print("Number of duplicate samples in x_train:", num_duplicates)
```

```
Checking for missing values...
Missing values in x_train: 0
Missing values in x_test: 0
Missing values in y_train: 0
Missing values in y_test: 0

Checking for duplicate images in x_train...
Number of duplicate samples in x_train: 0
```

5.Visualize a Few Features

```
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.datasets import mnist

# Load MNIST data
(x_train, y_train), (_, _) = mnist.load_data()

# Select a few random indices
num_images = 9
indices = np.random.choice(len(x_train), num_images, replace=False)

# Plot the selected images
plt.figure(figsize=(8, 8))
for i, idx in enumerate(indices):
    plt.subplot(3, 3, i + 1)
    plt.imshow(x_train[idx], cmap='gray')
    plt.title(f"Label: {y_train[idx]}")
    plt.axis('off')
plt.suptitle("Sample Handwritten Digits from MNIST", fontsize=16)
plt.tight_layout()
plt.show()
```



6.Identify Target and Features

```
from tensorflow.keras.datasets import mnist

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Features: pixel values of images
# Targets: the actual digit labels

print("Feature shape (x_train):", x_train.shape)
print("Target shape (y_train):", y_train.shape)

# Example feature (image) and target (label)
print("\nExample image (as array):")
print(x_train[0])
print("Corresponding label (target):", y_train[0])
```

```
Feature shape (x_train): (60000, 28, 28)
Target shape (y_train): (60000,)

Example image (as array):
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
```

```
 175  26 166 255 247 127   0   0   0   0]
[  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
 225 172 253 242 195  64   0   0   0   0]
[  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
  93  82  82  56  39   0   0   0   0   0]
[  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
  25   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
 150  27   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
 253 187   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
 253 249  64   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
 250 182   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
  78   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

7.Convert Categorical Columns to Numerical

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist

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

# One-hot encode the target labels
y_train_encoded = to_categorical(y_train, num_classes=10)
y_test_encoded = to_categorical(y_test, num_classes=10)

# Show example
print("Original label:", y_train[0])
print("One-hot encoded label:", y_train_encoded[0])
```

```
Original label: 5
One-hot encoded label: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

8.One-Hot Encoding

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

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

# Perform one-hot encoding on the labels
y_train_encoded = to_categorical(y_train, num_classes=10)
y_test_encoded = to_categorical(y_test, num_classes=10)

# Print shapes to verify
print("Shape of y_train before encoding:", y_train.shape)
print("Shape of y_train after encoding:", y_train_encoded.shape)
```

```
# Show an example
print("Original label:", y_train[0])
print("One-hot encoded label:", y_train_encoded[0])
```

```
Shape of y_train before encoding: (60000,)
Shape of y_train after encoding: (60000, 10)
Original label: 5
One-hot encoded label: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

9.Feature Scaling

```
from tensorflow.keras.datasets import mnist

# Load the data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Scale pixel values to the range [0, 1]
x_train_scaled = x_train.astype("float32") / 255.0
x_test_scaled = x_test.astype("float32") / 255.0

# Reshape for CNN input (if needed)
x_train_scaled = x_train_scaled.reshape(-1, 28, 28, 1)
x_test_scaled = x_test_scaled.reshape(-1, 28, 28, 1)

# Check result
print("Scaled pixel range (train):", x_train_scaled.min(), "to", x_train_scaled.max())
print("New shape of x_train:", x_train_scaled.shape)
```

```
Scaled pixel range (train): 0.0 to 1.0
New shape of x_train: (60000, 28, 28, 1)
```

10.Train-Test Split

```
from tensorflow.keras.datasets import mnist
from sklearn.model_selection import train_test_split

# Load the MNIST dataset
(x_train_full, y_train_full), (x_test, y_test) = mnist.load_data()

# Normalize pixel values
x_train_full = x_train_full.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# Reshape for CNN input
x_train_full = x_train_full.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# One-hot encode the labels
from tensorflow.keras.utils import to_categorical
y_train_full = to_categorical(y_train_full, 10)
y_test = to_categorical(y_test, 10)

# Split training set into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(
    x_train_full, y_train_full, test_size=0.2, random_state=42
)

# Confirm the shapes
print("Training set shape:", x_train.shape)
print("Validation set shape:", x_val.shape)
print("Test set shape:", x_test.shape)
```

```
Training set shape: (48000, 28, 28, 1)
Validation set shape: (12000, 28, 28, 1)
Test set shape: (10000, 28, 28, 1)
```

11.Evaluation

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam

# Build the CNN model
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'),
    Dense(10, activation='softmax')  # 10 classes (digits 0-9)
])

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model (assuming you have x_train, y_train, x_val, and y_val from the previous steps)
model.fit(x_train, y_train, epochs=10, validation_data=(x_val, y_val))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

print(f"Test accuracy: {test_acc * 100:.2f}%")
print(f"Test loss: {test_loss:.4f}")
```

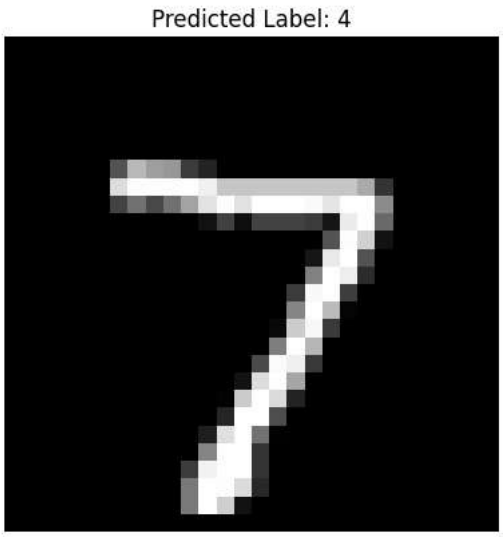
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 43s 27ms/step - accuracy: 0.8994 - loss: 0.3341 - val_accuracy: 0.9797 - val_loss: 0.0636
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 43s 29ms/step - accuracy: 0.9847 - loss: 0.0471 - val_accuracy: 0.9887 - val_loss: 0.0370
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 79s 27ms/step - accuracy: 0.9914 - loss: 0.0281 - val_accuracy: 0.9876 - val_loss: 0.0402
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 43s 29ms/step - accuracy: 0.9934 - loss: 0.0201 - val_accuracy: 0.9895 - val_loss: 0.0358
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 79s 27ms/step - accuracy: 0.9954 - loss: 0.0140 - val_accuracy: 0.9891 - val_loss: 0.0400
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 43s 29ms/step - accuracy: 0.9954 - loss: 0.0135 - val_accuracy: 0.9900 - val_loss: 0.0367
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 43s 29ms/step - accuracy: 0.9975 - loss: 0.0088 - val_accuracy: 0.9912 - val_loss: 0.0392
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 79s 27ms/step - accuracy: 0.9979 - loss: 0.0069 - val_accuracy: 0.9908 - val_loss: 0.0372
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 41s 27ms/step - accuracy: 0.9968 - loss: 0.0076 - val_accuracy: 0.9912 - val_loss: 0.0379
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 43s 29ms/step - accuracy: 0.9981 - loss: 0.0051 - val_accuracy: 0.9891 - val_loss: 0.0458
313/313 - 2s - 8ms/step - accuracy: 0.9909 - loss: 0.0362
Test accuracy: 99.09%
Test loss: 0.0362
```

12.Make Predictions from New Input

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist

# Load the trained model
# If you have a pre-trained model saved, you can load it like this:
# model = keras.models.load_model('path_to_your_model.h5')

# Build the CNN model (for context)
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'),
    Dense(10, activation='softmax')  # 10 output units for 10 classes (digits 0-9)
])
```

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Load MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize and reshape the input for CNN model
x_test = x_test.astype("float32") / 255.0
x_test = x_test.reshape(-1, 28, 28, 1)

# Example: Predict on a new image from the test set
new_input = x_test[0]  # Pick the first test image for this example
new_input = new_input.reshape(1, 28, 28, 1)  # Reshape to (1, 28, 28, 1)

# Make prediction
prediction = model.predict(new_input)
predicted_label = np.argmax(prediction)  # The label with the highest probability

# Display the input image and predicted label
plt.imshow(x_test[0], cmap='gray')
plt.title(f"Predicted Label: {predicted_label}")
plt.axis('off')
plt.show()

print(f"Predicted label: {predicted_label}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
```




```
Predicted label: 4
```

13.Convert to DataFrame and Encode

```
import pandas as pd
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Flatten the images into one-dimensional arrays
x_train_flattened = x_train.reshape(x_train.shape[0], -1)  # Flatten each image to a vector
x_test_flattened = x_test.reshape(x_test.shape[0], -1)  # Flatten each image to a vector

# Convert to DataFrame
train_df = pd.DataFrame(x_train_flattened)
test_df = pd.DataFrame(x_test_flattened)

# Add labels (y_train) to the DataFrame
train_df['label'] = y_train
test_df['label'] = y_test

# Show the first few rows of the DataFrame
print("Train DataFrame head:\n", train_df.head())
```

```
# One-hot encode the labels (convert numerical labels into binary format)
y_train_encoded = to_categorical(y_train, num_classes=10)
y_test_encoded = to_categorical(y_test, num_classes=10)

# Show the one-hot encoded labels for the first training example
print("\nOne-hot encoded label for the first training example:", y_train_encoded[0])
```

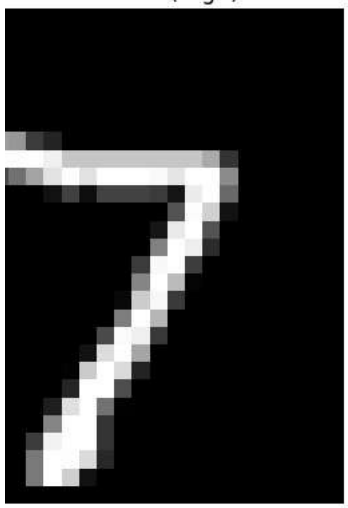
```
Train DataFrame head:
   0  1  2  3  4  5  6  7  8  9  ...  775  776  777  778  779  780  781  782  \
0  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
1  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
2  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
3  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
4  0  0  0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0

   783  label
0    0      5
1    0      0
2    0      4
3    0      1
4    0      9

[5 rows x 785 columns]

One-hot encoded label for the first training example: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

14.Predict the Final Grade

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize and reshape the images for CNN input
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# One-hot encode the 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)

# Define the CNN model
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'),
    Dense(10, activation='softmax')  # 10 classes (digits 0–9)
])

# Compile and train the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train_encoded, epochs=5, validation_data=(x_test, y_test_encoded))

# Predict on a new image (use the first image from the test set for simplicity)
new_input = x_test[0]  # Pick the first image for this example
new_input = new_input.reshape(1, 28, 28, 1)  # Reshape to (1, 28, 28, 1)

# Make prediction
prediction = model.predict(new_input)
predicted_label = np.argmax(prediction)  # The class with the highest probability

# Show the input image and predicted label
plt.imshow(x_test[0], cmap='gray')
plt.title(f"Predicted Final Grade (Digit): {predicted_label}")
plt.axis('off')
```

```
plt.show()

print(f"Predicted Final Grade: {predicted_label}")
```

```
/thon3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input
__(activity_regularizer=activity_regularizer, **kwargs)

━━━━━━━━━━━━━━━━━━━━ 54s 28ms/step - accuracy: 0.9160 - loss: 0.2830 - val_accuracy: 0.9867 - val_loss: 0.0397

━━━━━━━━━━━━━━━━━━━━ 80s 27ms/step - accuracy: 0.9860 - loss: 0.0441 - val_accuracy: 0.9891 - val_loss: 0.0316

━━━━━━━━━━━━━━━━━━━━ 81s 27ms/step - accuracy: 0.9909 - loss: 0.0281 - val_accuracy: 0.9912 - val_loss: 0.0261

━━━━━━━━━━━━━━━━━━━━ 83s 28ms/step - accuracy: 0.9943 - loss: 0.0186 - val_accuracy: 0.9900 - val_loss: 0.0312

━━━━━━━━━━━━━━━━━━━━ 81s 27ms/step - accuracy: 0.9956 - loss: 0.0136 - val_accuracy: 0.9895 - val_loss: 0.0361
━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
ed Final Grade (Digit): 7
```



```
Grade: 7
```

15.Deployment-Building an Interactive App

```
pip install streamlit tensorflow matplotlib
```

```
Collecting streamlit
  Downloading streamlit-1.45.0-py3-none-any.whl.metadata (8.9 kB)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.5.0)
Requirement already satisfied: blinker<2,>=1.5.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (1.9.0)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.5.2)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (8.1.8)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.0.2)
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.11/dist-packages (from streamlit) (24.2)
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.2.2)
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (11.2.1)
Requirement already satisfied: protobuf<7,>=3.20 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.29.4)
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (18.1.0)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.32.3)
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (9.1.2)
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.11/dist-packages (from streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (4.13.2)
Collecting watchdog<7,>=2.1.5 (from streamlit)
  Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl.metadata (44 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.3/44.3 kB 1.7 MB/s eta 0:00:00
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in /usr/local/lib/python3.11/dist-packages (from streamlit) (3.1.44)
Collecting pydeck<1,>=0.8.0b4 (from streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local/lib/python3.11/dist-packages (from streamlit) (6.4.2)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
```

```
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.3
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (3.1.6)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (4.23.0)
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (1.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0.
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.11/dist-packages (from gitpython!=3.1.19,<4,>=3.0.7->streaml
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.9)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.15.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->streamlit) (2025.2)
```

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist

# ... (rest of the code from cell 14) ...

# Compile and train the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train_encoded, epochs=5, validation_data=(x_test, y_test_encoded))

# Save the trained model
model.save('mnist_model.h5') # Add this line to save the model

# ... (rest of the code from cell 14) ...
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 52s 27ms/step - accuracy: 0.9958 - loss: 0.0127 - val_accuracy: 0.9913 - val_loss: 0.0276
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 26ms/step - accuracy: 0.9977 - loss: 0.0075 - val_accuracy: 0.9902 - val_loss: 0.0310
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 26ms/step - accuracy: 0.9978 - loss: 0.0071 - val_accuracy: 0.9897 - val_loss: 0.0411
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 27ms/step - accuracy: 0.9970 - loss: 0.0086 - val_accuracy: 0.9924 - val_loss: 0.0282
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 26ms/step - accuracy: 0.9978 - loss: 0.0062 - val_accuracy: 0.9912 - val_loss: 0.0332
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

16.Create a Prediction Function

```
import numpy as np
from tensorflow.keras.models import load_model
from PIL import Image, ImageOps

# Load your pre-trained model (you must have already saved it)
model = load_model("mnist_model.h5")

def predict_digit(image_input):
    """
    Predicts the digit from a 28x28 grayscale image using a trained CNN model.

    Parameters:
        image_input (PIL.Image or numpy.ndarray): The input image to predict.

    Returns:
        int: Predicted digit (0–9)
        np.ndarray: Array of prediction probabilities for each digit
    """
    # Convert input to a NumPy array if it's a PIL Image
    if isinstance(image_input, Image.Image):
```

```
        image = image_input.convert("L")  # Convert to grayscale
        image = ImageOps.invert(image)    # Invert colors to match MNIST
        image = image.resize((28, 28))    # Resize to 28x28 pixels
        img_array = np.array(image).astype("float32") / 255.0
    elif isinstance(image_input, np.ndarray):
        img_array = image_input.astype("float32") / 255.0
        if img_array.shape != (28, 28):
            raise ValueError("NumPy image array must be of shape (28, 28)")
    else:
        raise TypeError("Input must be a PIL.Image or numpy.ndarray.")

    # Reshape to fit model input: (1, 28, 28, 1)
    img_array = img_array.reshape(1, 28, 28, 1)

    # Predict
    prediction = model.predict(img_array)
    predicted_label = int(np.argmax(prediction))

    return predicted_label, prediction[0]
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
```