```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalize images
X_train = X_train / 255.0
X_test = X_test / 255.0

# Reshape to add channel dimension
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

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

```
# Build the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Dropout(0.3),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.3),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.4),
    Dense(10, activation='softmax')  # 10 classes for digits (0-9)
])

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

```
/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)
```

```
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 48s 31ms/step - accuracy: 0.8345 - loss: 0.5107 - val_accuracy: 0.9798 - val_loss: 0.0687
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 45s 30ms/step - accuracy: 0.9670 - loss: 0.1096 - val_accuracy: 0.9858 - val_loss: 0.0466
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9785 - loss: 0.0746 - val_accuracy: 0.9884 - val_loss: 0.0404
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9806 - loss: 0.0641 - val_accuracy: 0.9898 - val_loss: 0.0342
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 81s 30ms/step - accuracy: 0.9819 - loss: 0.0554 - val_accuracy: 0.9896 - val_loss: 0.0342
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 48s 32ms/step - accuracy: 0.9838 - loss: 0.0518 - val_accuracy: 0.9908 - val_loss: 0.0301
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 81s 31ms/step - accuracy: 0.9851 - loss: 0.0484 - val_accuracy: 0.9903 - val_loss: 0.0336
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 80s 30ms/step - accuracy: 0.9856 - loss: 0.0452 - val_accuracy: 0.9898 - val_loss: 0.0328
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 85s 32ms/step - accuracy: 0.9870 - loss: 0.0405 - val_accuracy: 0.9913 - val_loss: 0.0328
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 80s 30ms/step - accuracy: 0.9887 - loss: 0.0372 - val_accuracy: 0.9930 - val_loss: 0.0281
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9902 - loss: 0.0266
Test Accuracy: 0.99
```

```
# Save the trained model
model.save("handwritten_character_recognition.h5")
```

```
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
```

handwritten digits prediction using deep learning

```
from sklearn.datasets import fetch_openml
mnist=fetch_openml("mnist_784")
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Load the MNIST dataset
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target

# Ensure that the data is in the correct format
X = X.astype(np.float32)  # Convert pixel values to float32
y = y.astype(np.int8)     # Convert target values to integers

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize the MLP classifier
mlp = MLPClassifier(hidden_layer_sizes=(64, 64),
                    max_iter=50,
                    alpha=1e-4,
                    solver='adam',
                    random_state=42,
                    verbose=10,
                    tol=1e-4)

# Train the model
mlp.fit(X_train, y_train)

# Predict on the test data
y_pred = mlp.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Visualize some predictions
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for ax, image, prediction, true_label in zip(axes.ravel(), X_test.to_numpy(), y_pred, y_test):
    ax.set_axis_off()
    image = image.reshape(28, 28)
    ax.imshow(image, cmap='gray')
    ax.set_title(f"Pred: {prediction}, True: {true_label}")

plt.tight_layout()
plt.show()
```

```
Iteration 1, loss = 2.10292369
Iteration 2, loss = 0.63468546
Iteration 3, loss = 0.41170029
Iteration 4, loss = 0.32570292
Iteration 5, loss = 0.27403755
Iteration 6, loss = 0.24147559
Iteration 7, loss = 0.21699058
Iteration 8, loss = 0.19487723
Iteration 9, loss = 0.17771913
Iteration 10, loss = 0.16357822
Iteration 11, loss = 0.15891865
Iteration 12, loss = 0.14965729
Iteration 13, loss = 0.12972488
Iteration 14, loss = 0.13554853
Iteration 15, loss = 0.13080247
Iteration 16, loss = 0.13119637
Iteration 17, loss = 0.11904129
Iteration 18, loss = 0.11818511
Iteration 19, loss = 0.11466374
Iteration 20, loss = 0.11052036
Iteration 21, loss = 0.10766985
Iteration 22, loss = 0.10104356
Iteration 23, loss = 0.09436340
Iteration 24, loss = 0.08939378
Iteration 25, loss = 0.09473639
Iteration 26, loss = 0.08972806
Iteration 27, loss = 0.08268881
Iteration 28, loss = 0.08483314
Iteration 29, loss = 0.08084884
Iteration 30, loss = 0.07425225
Iteration 31, loss = 0.06775623
Iteration 32, loss = 0.06304130
Iteration 33, loss = 0.07305904
Iteration 34, loss = 0.06126501
Iteration 35, loss = 0.05835775
Iteration 36, loss = 0.06709495
Iteration 37, loss = 0.06276928
Iteration 38, loss = 0.05704420
Iteration 39, loss = 0.05660724
Iteration 40, loss = 0.04841852
Iteration 41, loss = 0.05089388
Iteration 42, loss = 0.05344457
Iteration 43, loss = 0.05155276
Iteration 44, loss = 0.04803985
Iteration 45, loss = 0.04400076
Iteration 46, loss = 0.04093383
Iteration 47, loss = 0.03991760
Iteration 48, loss = 0.03962692
Iteration 49, loss = 0.04080373
Iteration 50, loss = 0.04110693
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1343
           1       0.98      0.98      0.98      1600
           2       0.96      0.94      0.95      1380
           3       0.93      0.95      0.94      1433
           4       0.94      0.97      0.96      1295
           5       0.97      0.93      0.95      1273
           6       0.98      0.97      0.98      1396
           7       0.95      0.97      0.96      1503
           8       0.94      0.93      0.94      1357
           9       0.95      0.94      0.94      1420

    accuracy                           0.96     14000
   macro avg       0.96      0.96      0.96     14000
weighted avg       0.96      0.96      0.96     14000

Confusion Matrix:
[[1314    1    5    2    2    1    2    3   10    3]
 [   0 1565    3    3    1    1    3    7   12    5]
 [   4    8 1304   12    8    3    3   24   11    3]
 [   0    2   12 1364    2   14    0    9   12   18]
 [   2    1    2    0 1260    0    3   10    3   14]
 [   3    6    5   41    4 1178   12    4   15    5]
 [   6    5    2    2   11    8 1356    1    3    2]
 [   4    2    8    2   11    1    0 1458    2   15]
 [   2    9   17   29    5    9    2    8 1266   10]
 [   8    2    3    9   36    4    1   17    9 1331]]
Accuracy: 95.69%
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimize
  warnings.warn(
```