Handwritten Digit Recognition

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

- import os: This imports the os module which provides functions for interacting with the operating system.
- import cv2: This imports the OpenCV library which is used for image processing tasks.
- import numpy as np: This imports the NumPy library under the alias np. NumPy is used for numerical computing in Python.
- import matplotlib.pyplot as plt: This imports the pyplot module from the Matplotlib library, which is used for plotting and visualizations.
- import tensorflow as tf: This imports TensorFlow, a popular deep learning framework.

```
    mnist = tf.keras.datasets.mnist
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

- **mnist = tf.keras.datasets.mnist**: This loads the MNIST dataset from the Keras datasets module. MNIST is a dataset of handwritten digits commonly used for training image classification models.
- **(x_train, y_train), (x_test, y_test) = mnist.load_data()**: This line loads the MNIST dataset into training and testing sets, where **x_train** and **x_test** contain the images of handwritten digits, and **y_train** and **y_test** contain the corresponding labels.

```
    x_train = tf.keras.utils.normalize(x_train, axis=1)
    x_test = tf.keras.utils.normalize(x_test, axis=1)
```

These lines normalize the pixel values of the images in the training and testing sets. Normalization is a common preprocessing step in machine learning where the data is scaled to have a mean of 0 and a standard deviation of 1 along a particular axis (in this case, along the axis representing the image dimensions).

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

This code defines a sequential model using Keras' Sequential API. The model consists of:

- A Flatten layer: This layer flattens the input image into a 1D array. The input shape **(28, 28)** represents the dimensions of the input images.

- Two Dense layers with 128 units each and ReLU activation: These layers are fully connected (dense) layers with 128 neurons each and ReLU activation function, which introduces non-linearity into the model.
- A Dense layer with 10 units and softmax activation: This is the output layer with 10 units corresponding to the 10 possible classes (digits 0 through 9) in the MNIST dataset. Softmax activation is used to obtain probabilities for each class.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

This line compiles the model, specifying the optimizer, loss function, and metrics to be used during training. In this case:

- optimizer='adam': Adam is a popular optimization algorithm used for training neural networks.
- loss='sparse_categorical_crossentropy': This is the loss function used for categorical classification problems where the labels are integers.
- metrics=['accuracy']: During training, the model will track accuracy as a metric.

```
model.fit(x_train, y_train, epochs=10)
```

This line trains the model on the training data for 10 epochs. During training, the model learns to map input images to their corresponding labels.

```
model.save('handwritten.h5')
```

This line saves the trained model to a file named `'handwritten.h5'` in HDF5 format. This format is commonly used for saving and loading Keras models.

```
model = tf.keras.models.load_model('handwritten.h5')
```

This line loads the saved model from the file `'handwritten.h5'` into the variable `model`. Now the model is ready for making predictions.

```
image_number = 1
while os.path.isfile(f"digits/digit{image_number}.png"):
    try:
        img = cv2.imread(f"digits/digit{image_number}.png")[:, :, 0]
        img = np.invert(np.array([img]))
        prediction = model.predict(img)
        print(f"The number is probably a {np.argmax(prediction)}")
        plt.imshow(img[0], cmap=plt.cm.binary)
        plt.show()  # Display the image
    except Exception as e:
        print(f'Error processing image {image_number}: {str(e)}')
    finally:
        image_number += 1
```

This block of code iterates over images of handwritten digits stored in the `digits` directory. It loads each image using OpenCV (`cv2.imread`), converts it to grayscale, inverts the pixel values, and then uses the trained model to predict the digit. The predicted digit is printed, and the image is displayed using Matplotlib. The loop continues until there are no more image files found in the directory. Any errors that occur during image processing are caught and printed.