Handwritten Digit Recognition Code Documentation

1. Data Loading :

    The code loads the MNIST dataset, which consists of handwritten digit images. The dataset is split into training and testing sets, where `X_train` contains the images and `y_train` contains the corresponding labels for training, and `X_test` and `y_test` for testing.

2. Data Preprocessing:

    Images are normalized to the range [0, 1] using the `normalize` function. Normalization is crucial for ensuring that the input features have similar scales, which can improve the convergence and performance of machine learning algorithms.

    Labels are converted into one-hot encoded vectors using the `to_categorical` function. One-hot encoding is essential for classification tasks, as it transforms categorical labels into a format that is suitable for training machine learning models.

3. Exploratory Data Analysis (EDA):

    The code displays a few random images from the training set using matplotlib. This step helps in visually inspecting the data and understanding its characteristics.

    It plots the distribution of different digits in the training dataset using a bar chart. Understanding the distribution of classes in the dataset is essential for assessing class balance and potential biases in the data .

    Machine learning models trained on a balanced dataset have an equal opportunity to learn patterns and features from each class. This helps prevent biases toward specific classes and ensures that the model learns to generalize well across all classes.

3. Model Building and Evaluation:

Logistic Regression Model:

    A logistic regression model (`lg_model`) is instantiated with default parameters. Logistic regression is a simple yet effective algorithm for binary classification tasks.

The model is trained on the training data `(X_train, y_train)` using the `fit` method. Training involves adjusting the model parameters to minimize the logistic loss function.

Predictions are made on the test set (`X_test`) using the `predict` method. The accuracy of the logistic regression model is computed using the `accuracy_score` function from the `metrics` module.

Confusion matrix and classification report for the logistic regression model are printed. These metrics provide insights into the model's performance and help in identifying potential areas for improvement.

K-Nearest Neighbors (KNN) Model:

A KNN classifier (`knn_model`) is instantiated with `n_neighbors=3`, indicating the number of neighbors to consider during classification.

The model is trained on the training data `(X_train, y_train)` using the `fit` method. KNN is a non-parametric algorithm that stores all training data points and classifies new instances based on their proximity to existing data points.

Predictions are made on the test set (`X_test`) using the `predict` method. The accuracy of the KNN model is computed using the `accuracy_score` function.

Confusion matrix and classification report for the KNN model are printed, providing insights into its performance and potential limitations.

Convolutional Neural Network (CNN) Model:

The data is reshaped to fit the CNN architecture. CNNs require input data in a specific format `(num_samples, width, height, channels).`

The CNN model (`cnn_model`) is defined using Keras Sequential API. It consists of convolutional and dense layers, which are commonly used in image classification tasks.

The model is compiled with categorical cross-entropy loss and Adam optimizer, which are standard choices for multiclass classification problems.

Training is performed on the training data `(X_train_cnn, y_train_one_hot)` using the `fit` method. The `batch_size` and `epochs` parameters control the number of samples processed per batch and the number of training iterations, respectively.

Accuracy of the CNN model is evaluated on the test set `(X_test_cnn, y_test_one_hot)` using the `evaluate` method. The `evaluate` method returns the loss value and evaluation metrics specified during model compilation.

Predictions are made for the test set, and the predicted class labels are obtained using `argmax` function. The true class labels are also extracted from one-hot encoded vectors.

Confusion matrix and classification report for the CNN model are printed, providing insights into its performance and potential areas for improvement.

4. Model Comparison:

The accuracies of all models are stored in a dictionary named `results`, where each key-value pair represents a model name and its corresponding accuracy.

A bar chart is plotted to compare the accuracies of different models. Visualization helps in easily comparing the performance of multiple models and identifying the best-performing one.

Conclusion:

The code successfully builds, trains, and evaluates multiple models for handwritten digit recognition.

It provides insights into the performance of each model and facilitates model comparison.

The documentation enables clear understanding and reproducibility of the experiment.