**Neural Network Model for MNIST Digit Classification**

Here the step by step explanation of code file:

1. **Importing Libraries:**
   * The necessary libraries for the code are imported, including TensorFlow, Keras, Matplotlib, and NumPy.
2. **Loading and Preparing the Dataset:**
   * The MNIST dataset is loaded using the **keras.datasets.mnist.load\_data()** function.
   * The dataset is split into training and testing sets, stored in **X\_train** and **Y\_train** for training data, and **X\_test** and **Y\_test** for testing data.
   * The shapes of the training data (**X\_train**) and the data at index 0 (**X\_train[0]**) are displayed using the **.shape** attribute.
   * The image at index 0 (**X\_train[0]**) and index 1 (**X\_train[1]**) are displayed using **plt.matshow()** to visualize the images.
   * The corresponding labels at index 0 and 1 (**Y\_train[0]** and **Y\_train[1]**) are displayed.
3. **Data Preprocessing:**
   * The pixel values of the images in the training and testing sets are normalized by dividing them by 255 to bring them within the range of 0 to 1.
   * The **X\_train** and **X\_test** arrays are flattened using the **.reshape()** method, converting the 2D arrays into 1D arrays, where each image is represented as a 1D array of length 784 (28x28 pixels).
   * The shapes of the flattened training and testing sets (**X\_train\_flatten.shape** and **X\_test\_flatten.shape**) are displayed.
4. **Building the Model:**
   * A sequential model is created using **Sequential()**.
5. **Convolutional Layers:**

* The first convolutional layer applies 32 filters, each of size 3x3, to the input image. It uses the ReLU activation function to introduce non-linearity.
* A max pooling layer follows the first convolutional layer. It reduces the spatial dimensions of the previous layer's output by taking the maximum value within each 2x2 window.

1. **Additional Convolutional and Pooling Layers:**

* Two more sets of convolutional and max pooling layers are added, with an increasing number of filters (64 and 128, respectively). These layers extract higher-level features while reducing spatial dimensions.

1. **Flattening Layer:**

* A flattening layer is inserted to convert the multi-dimensional feature maps from the previous layer into a 1D vector. This prepares the data for the fully connected layers.

1. **Fully Connected Layers:**

* Two fully connected layers are added after the flattening layer.
* The first fully connected layer has 64 neurons and uses the ReLU activation function.
* The second fully connected layer has 10 neurons and uses the softmax activation function.
* The softmax function produces a probability distribution over the classes, indicating the likelihood of the input belonging to each class.

1. **Training the Model:**
   * The model is trained using the **fit()** function, passing the flattened training data (**X\_train\_flatten**) and corresponding labels (**Y\_train**).
   * The number of epochs is set to 7.
2. **Evaluating the Model:**
   * The model's performance is evaluated using the **evaluate()** function, passing the flattened testing data (**X\_test\_flatten**) and corresponding labels (**Y\_test**).
   * The evaluation provides the loss value and accuracy of the model on the testing set.
3. **Making Predictions:**
   * The model's predictions are obtained using the **predict()** function, passing the flattened testing data (**X\_test\_flatten**).
   * The predicted values for the testing set are stored in **y\_predicted**.
4. **Visualizing Predictions:**
   * The original image at index 0 in the testing set (**X\_test[0]**) is displayed using **plt.matshow()**.
   * The predicted value for the image at index 0 is obtained using **np.argmax(y\_predicted[0])**, which returns the index of the highest probability value.