In lab Tasks:

Code outputs:

```
[1]  from keras.models import Sequential
     from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
     from keras.datasets import mnist
     from keras.utils import to_categorical
```

```
[2]  (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

     train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
     test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

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

```
[3]  train_labels = to_categorical(train_labels)
     test_labels = to_categorical(test_labels)
```

```
[5]  model = Sequential()
     model.add(Conv2D(32, (3,3), activation = 'relu', input_shape=(28,28,1)))
     model.add(MaxPooling2D((2,2)))
     model.add(Conv2D(32, (3,3), activation = 'relu'))
     model.add(MaxPooling2D((2,2)))
     model.add(Flatten())
     model.add(Dense(64, activation='relu'))
     model.add(Dense(10, activation='softmax'))
```

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

```
model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_data=(test_images, test_labels))

Epoch 1/5
938/938 [==============================] - 41s 42ms/step - loss: 0.2004 - accuracy: 0.9387 - val_loss: 0.0813 - val_accuracy: 0.9751
Epoch 2/5
938/938 [==============================] - 37s 39ms/step - loss: 0.0614 - accuracy: 0.9808 - val_loss: 0.0444 - val_accuracy: 0.9847
Epoch 3/5
938/938 [==============================] - 38s 41ms/step - loss: 0.0435 - accuracy: 0.9865 - val_loss: 0.0359 - val_accuracy: 0.9881
Epoch 4/5
938/938 [==============================] - 36s 38ms/step - loss: 0.0337 - accuracy: 0.9894 - val_loss: 0.0334 - val_accuracy: 0.9887
Epoch 5/5
938/938 [==============================] - 38s 41ms/step - loss: 0.0275 - accuracy: 0.9910 - val_loss: 0.0297 - val_accuracy: 0.9901
<keras.src.callbacks.History at 0x7deeb0a675b0>
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {test_acc}')

313/313 [==============================] - 2s 7ms/step - loss: 0.0297 - accuracy: 0.9901
Test Accuracy: 0.9901000261306763
```

```
[1]  from keras.models import Sequential
     from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
     from keras.datasets import mnist
     from keras.utils import to_categorical
```

Import Libraries: This section imports necessary functions and modules from the Keras library, such as tools for creating neural network models (`Sequential` model), layers (like convolutional and pooling layers), the MNIST dataset, and utilities for data processing.

Sample Image:

```
[2]  (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Loading MNIST Data: This line loads the MNIST dataset, specifically splitting it into training and testing sets. The training set consists of images with corresponding labels, and the test set is a separate collection of images and their labels used for evaluating the model.

```
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
```

Preparing Image Data: The images are reshaped to a format suitable for the model. They are originally 28x28 pixels but are reshaped into a 4D tensor (60000 samples for training, 10000 for testing) with dimensions (28, 28, 1). Additionally, the pixel values (which range from 0 to 255) are scaled to a range between 0 and 1 by dividing by 255, which helps the model learn better.

```
[3]  train_labels = to_categorical(train_labels)
     test_labels = to_categorical(test_labels)
```

One-Hot Encoding Labels: This step converts the categorical labels (numbers from 0 to 9) into a binary matrix representation called one-hot encoding. It helps the model understand the different categories without assuming any ordinal relationship between the numbers.

```
[5]  model = Sequential()
     model.add(Conv2D(32, (3,3), activation = 'relu', input_shape=(28,28,1)))
     model.add(MaxPooling2D((2,2)))
     model.add(Conv2D(32, (3,3), activation = 'relu'))
     model.add(MaxPooling2D((2,2)))
     model.add(Flatten())
     model.add(Dense(64, activation='relu'))
     model.add(Dense(10, activation='softmax'))
```

Building the Neural Network Model: This code constructs the neural network architecture using the Sequential API from Keras.

- It begins with a Convolutional Neural Network (CNN) consisting of convolutional layers (`Conv2D`) and max-pooling layers (`MaxPooling2D`), designed to learn features from the images.
- The Flatten layer transforms the 2D output from the convolutional layers into a 1D array, preparing it for the dense layers.
- Dense layers (`Dense`) are fully connected layers that perform computations and final classification. The last dense layer uses a 'softmax' activation function to output probabilities for each of the 10 classes (digits from 0 to 9).

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

Compiling the Model: This line configures the model for training by specifying the optimizer (Adam), the loss function (categorical cross-entropy, suitable for multi-class classification), and the metric to evaluate during training (accuracy).

```
odel.fit(train_images, train_labels, epochs=5, batch_size=64, validation_data=(test_images, test_labels))
```

Training the Model: This line trains the model using the training images and their labels. It iterates through the dataset five times (epochs=5) in batches of 64 images (batch_size=64). It also uses the test set for validation during training to monitor how well the model generalizes to unseen data.

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {test_acc}')
```

Evaluating the Model: This evaluates the trained model on the test set to assess its performance. It calculates the loss and accuracy of the model on the unseen test data and prints the test accuracy.

`test_loss` stores the value of the loss obtained during the evaluation on the test data.
`test_acc` stores the accuracy achieved by the model on the test dataset.