In-Lab:

Lab Task 1:

```
# Import necessary libraries
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.datasets import mnist
from keras.utils import to_categorical
```

Explanation:

Import the necessary libraries for building and training a convolutional neural network (CNN) model. These libraries include Keras, which provides a high-level API for building neural networks, and the MNIST dataset, which contains images of handwritten digits used for training and evaluating the model.

Lab Task 2:

```
# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
# Reshape and normalize the images
train_images = train_images.reshape((60000, 28, 28,
1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28,
1)).astype('float32') / 255
```

Explanation:

Load the MNIST dataset using the provided Keras function. Preprocess the images by reshaping them into a format suitable for the CNN model and normalizing their pixel values between 0 and 1. This step ensures consistent input data for the model.

Lab Task 3:

```
# One-hot encode the labels
train_labels = to_categorical (train_labels)
test_labels = to_categorical (test_labels)
```

Explanation:

One-hot encode the labels associated with each image in the dataset. This involves converting the categorical labels (digit classes) into binary vectors, where each position in the vector corresponds to a specific digit class. This representation is required for multi-class classification tasks.

Lab Task 4:

```
# Build the CNN model
model = Sequential()
# Step 1: Convolutional Layer with ReLU activation
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
1)))
# Step 2: Max Pooling Layer
model.add(MaxPooling2D((2, 2)))
# Step 3: Convolutional Layer with ReLU activation
model.add(Conv2D(64, (3, 3), activation='relu'))
# Step 4: Max Pooling Layer
model.add(MaxPooling2D((2, 2)))
# Step 5: Flatten Layer
model.add(Flatten())
# Step 6: Dense (Fully Connected) Layer with ReLU activation
model.add(Dense (64, activation='relu'))
# Step 7: Output Layer with Softmax activation (for multi-class
classification)
model.add(Dense (10, activation='softmax'))
```

Explanation:

Build the CNN model using a sequential model structure. The model consists of several layers, each performing a specific function in extracting features and classifying the input images:

- Convolutional Layer: Applies filters to extract local features from the input image.
- Max Pooling Layer: Reduces the spatial dimensions of the feature maps by selecting the maximum values in each pooling window. This helps in preventing overfitting.
- Flatten Layer: Converts the 2D feature maps into a 1D vector for further processing.
- Dense Layer: Performs fully connected operations to combine the extracted features and make predictions.

Lab Task 5:

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

Explanation:

Compile the CNN model by specifying the optimizer, loss function, and metrics to be used during training. The optimizer determines the algorithm used to update the model's weights, the loss function measures the model's error on the training data, and the metrics provide additional insights into the model's performance.

Lab Task 6:

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

test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {round(test_acc,4)}')
```

Explanation:

Train the CNN model using the compiled configuration. Train the model for a specified number of epochs (iterations over the entire training dataset) and batch size (number of samples processed per update). Evaluate the trained model's performance on the test dataset and calculate its accuracy.

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
938/938 [==============================] - 59s 62ms/step - loss: 0.1709 - accuracy: 0.9492 - val_loss: 0.0608 - val_accuracy: 0.9811
Epoch 2/5
938/938 [==============================] - 57s 60ms/step - loss: 0.0520 - accuracy: 0.9839 - val_loss: 0.0424 - val_accuracy: 0.9853
Epoch 3/5
938/938 [==============================] - 53s 57ms/step - loss: 0.0371 - accuracy: 0.9886 - val_loss: 0.0306 - val_accuracy: 0.9896
Epoch 4/5
938/938 [==============================] - 56s 60ms/step - loss: 0.0284 - accuracy: 0.9911 - val_loss: 0.0323 - val_accuracy: 0.9898
Epoch 5/5
938/938 [==============================] - 63s 67ms/step - loss: 0.0232 - accuracy: 0.9928 - val_loss: 0.0276 - val_accuracy: 0.9897
313/313 [==============================] - 3s 9ms/step - loss: 0.0276 - accuracy: 0.9897
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0276 - accuracy: 0.9897
Test Accuracy: 0.9897
```