**In-Lab**

**In-Lab Task 1**

|  |
| --- |
| **Code:**  # 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  print("Libraries Imported Successfully!")  **Output:**    **Explanation:**  This task involved importing the necessary libraries for building and training a CNN model for image classification using Keras. These libraries provide the essential tools for constructing neural networks, preprocessing data, and evaluating model performance. |

**In-Lab Task 2**

|  |
| --- |
| **Code:**  # 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  print("Loaded and processed the dataset. \nTraining and testing images reshaped and normalized successfully!")  **Output:**    **Explanation:**  The MNIST dataset, containing images of handwritten digits, was loaded and preprocessed in this task. Preprocessing involved reshaping the images into a format suitable for the CNN model and normalizing the pixel values to lie between 0 and 1. This ensures that the model's training process is not affected by varying image brightness or contrast. |

**In-Lab Task 3**

|  |
| --- |
| **Code:**  # One-hot encode the labels  train\_labels = to\_categorical (train\_labels)  test\_labels = to\_categorical (test\_labels)  print("Labels One-hot encoded successfully!")  **Output:**    **Explanation:**  One-hot encoding was applied to the labels associated with each image in the MNIST dataset. This process converts the categorical labels into numerical vectors, where each vector has 10 elements, one for each possible digit (0-9). This representation makes it easier for the CNN model to learn the relationship between the images and their corresponding digits. |

**In-Lab Task 4**

|  |
| --- |
| **Code:**  # 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'))  print("Built the CNN model successfully!")  **Output:**    **Explanation:**   * **The CNN model was constructed using the** **Keras Sequential API.** This API provides a straightforward way to define and build neural network models layer by layer. The model consists of seven layers: * **Convolutional layer with ReLU activation** ***(32 Layers)***: This layer extracts features from the input images using convolutional filters. ReLU activation adds non-linearity to the model's decision-making process. * **Max pooling layer**: This layer reduces the dimensionality of the feature maps by selecting the maximum value in each region. This helps in preventing overfitting and improving computational efficiency. * **Convolutional layer with ReLU activation** ***(64 Layers)***: This layer applies another layer of feature extraction and non-linearity. * **Max pooling layer:** Another round of dimensionality reduction using max pooling. * **Flatten layer:** This layer converts the multidimensional feature maps into a one-dimensional vector, preparing the data for the fully connected layers. * **Fully connected layer with ReLU activation**: This layer performs high-level feature extraction and decision-making, connecting the flatten layer to the output layer. * **Output layer with softmax activation**: The final layer produces a probability distribution over the 10 possible digits (0-9) for each input image. Softmax activation ensures the probabilities sum to 1. |

**In-Lab Task 5**

|  |
| --- |
| **Code:**  # Compile the model  model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  print("Compilation of the model completed successfully!")  **Output:**  **Explanation:**  The CNN model was compiled using the Adam optimizer, categorical cross-entropy loss function, and accuracy metric. The Adam optimizer is a popular choice for training neural networks due to its efficient optimization of model parameters. Categorical cross-entropy is an appropriate loss function for multi-class classification tasks, measuring the difference between the predicted and actual probability distributions. The accuracy metric evaluates the model's ability to correctly classify the handwritten digits. |

**In-Lab Task 6**

|  |
| --- |
| **Code:**  # Train the model  model.fit(train\_images,train\_labels,epochs=5,batch\_size=64,validation\_data=(test\_images,test\_labels))  print("Model trained successfully!")  **Output:**  **Explanation:**  The CNN model was trained on the preprocessed training data for 5 epochs using batches of 64 images. Each epoch involves presenting the entire training dataset to the model multiple times. Batches divide the training data into smaller chunks, allowing for efficient training and preventing memory overload. During training, the model's performance was evaluated on the validation data, providing insights into its generalization ability. |

**In-Lab Task 7**

|  |
| --- |
| **Code:**  # Evaluate the model on the test set  test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)  print(f"Test Accuracy: {round(test\_acc,4)}")  print("Model Evaluation Completed!\n Every step has been completed!")  **Output:**  **Explanation:**  The trained CNN model was evaluated on the test data, a set of unseen images, to assess its generalization performance in classifying handwritten digits. The accuracy of the model was reported, indicating its ability to correctly identify the digits in the test set. This evaluation provides an unbiased assessment of the model's real-world performance. |