**ASSESMENT-1**

**PROBLEM STATEMENT-** Setting up the Spyder IDE Environment and Executing a Python Program

**SOURCE CODE:**

1. Visit the Spyder Website: Open your web browser and go to the Spyder IDE website. The official website for Spyder is <https://www.spyder-ide.org/>.
2. Download Spyder: On the Spyder IDE website, you should see a prominent download button. Click on it to navigate to the download page. Choose the appropriate download option based on your operating system (Windows, macOS, or Linux).
3. Download the Installer: Once you've selected your operating system, click on the download button to download the installer file. The file you download will typically be an executable file for Windows (.exe), a disk image for macOS (.dmg), or an installation script for Linux (.sh).
4. Run the Installer: Locate the downloaded installer file on your computer and run it by double-clicking on it. Follow the instructions provided by the installer to install Spyder on your system. The installation process may vary slightly depending on your operating system.
5. Launch Spyder: After the installation is complete, you can launch Spyder by searching for it in your system's applications or by clicking on the shortcut created during the installation process.
6. Optional: Install Additional Dependencies: Depending on your usage, you may need to install additional Python packages or libraries for specific functionalities. You can use pip to install these packages as needed.
7. Update Spyder (optional): Similar to the pip installation method, you can periodically update Spyder to get the latest features and bug fixes. Check for updates on the Spyder website and follow the instructions provided to update your installation.

That's it! You have now installed Spyder IDE on your system directly from the Spyder website. You can start using Spyder for Python development

**Installations:**

The command to Install:

**“pip install keras tensorflow”**

**File Name: program1.py**

# Importing necessary libraries

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Flatten

from keras.datasets import mnist

from keras.utils import to\_categorical

# Load and preprocess the MNIST dataset

(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

train\_labels = to\_categorical(train\_labels)

test\_labels = to\_categorical(test\_labels)

# Define the neural network model

model = Sequential()

model.add(Flatten(input\_shape=(28, 28, 1)))

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))

# Compile the model

model.compile(optimizer='adam',

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# Train the model

model.fit(train\_images, train\_labels, epochs=5, batch\_size=64, verbose=1)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

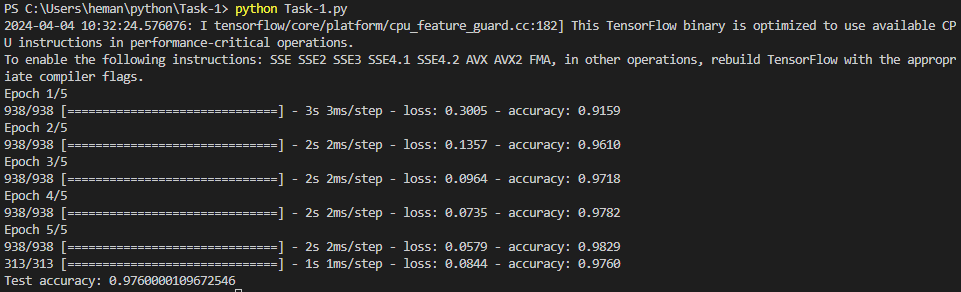
print('Test accuracy:', test\_acc)

**Explanation:**

1. **Importing Necessary Libraries:**
   * We import libraries such as numpy for numerical operations, Sequential and Dense from Keras for defining the neural network model, Flatten for flattening the input data, mnist for loading the MNIST dataset, and to\_categorical for one-hot encoding of labels.
2. **Load and Preprocess the MNIST Dataset:**
   * We load the MNIST dataset using the mnist.load\_data() function, which gives us training and testing images along with their corresponding labels.
   * The images are reshaped to the desired input shape (28x28 pixels) and normalized by dividing by 255 to scale the pixel values between 0 and 1.
   * The labels are one-hot encoded using to\_categorical to convert them into binary vectors.
3. **Define the Neural Network Model:**
   * We define a sequential model using Sequential(), which allows us to create a linear stack of layers.
   * The input layer is flattened using Flatten() to convert the 2D input images into a 1D array.
   * We add a dense (fully connected) layer with ReLU activation function and 128 neurons.
   * The output layer consists of 10 neurons (equal to the number of classes) with softmax activation for multi-class classification.
4. **Compile the Model:**
   * We compile the model using model.compile() with parameters such as optimizer (here, 'adam'), loss function ('categorical\_crossentropy' for multi-class classification), and evaluation metric ('accuracy').
5. **Train the Model:**
   * We train the model using model.fit() with training images and labels, specifying the number of epochs and batch size.
6. **Evaluate the Model:**
   * We evaluate the trained model on the test data using model.evaluate(), which computes the test loss and accuracy.
   * The test accuracy is printed to evaluate the performance of the model.

This code demonstrates a basic deep learning program using the Keras library to create a neural network for image classification.

**OUTPUT:**

****