TASK 1:

I have provided a code snippet that configures and trains a neural network for the MNIST dataset classification task. Here's an explanation of how I configured and trained the network and the reasons behind my choices:

Loading and Preprocessing Data:

I loaded the MNIST dataset using Keras's mnist.load_data() method, which splits the dataset into training and testing sets.

I normalized the pixel values of the images by dividing them by 255.0. Normalization is a common practice that scales the values to the range [0, 1], which can help improve training stability.

Model Architecture:

I defined a neural network model using Keras Sequential API.

The Flatten layer is used to flatten the 28x28 pixel images into a 1D array.

A dense hidden layer with 128 units and ReLU activation is added to capture complex patterns in the data.

The output layer consists of 10 units with softmax activation, suitable for multiclass classification. Softmax produces probability scores for each class.

The choice of a single hidden layer with 128 units is a reasonable starting point for the MNIST dataset, balancing simplicity and model capacity.

Model Compilation:

I compiled the model using the 'sparse_categorical_crossentropy' loss function. This loss function is suitable for multiclass classification tasks.

'Adam' optimizer was chosen, which is a popular and effective optimizer for various tasks.

The metric chosen for monitoring model performance is 'accuracy,' which is appropriate for classification tasks.

Training:

The model was trained using the fit method on the training data (X_train and y_train) for 10 epochs. The use of an epoch count of 10 is a common starting point for training neural networks and can be adjusted based on validation results.

I used a validation split of 20% (validation_split = 0.2) to monitor the model's performance during training.

Prediction and Evaluation:

I made predictions on the test data using the trained model.

I obtained class predictions by taking the index of the class with the highest probability.

I calculated the accuracy score using sklearn.metrics.accuracy_score to evaluate the model's performance on the test data.

Visualization:

I plotted the training loss and validation loss over epochs to monitor the training progress.

Similarly, I plotted the training accuracy and validation accuracy.

This code demonstrates a standard approach to training a neural network for image classification using TensorFlow and Keras. The choices I made regarding model architecture, loss function, optimizer, and evaluation metric are common and suitable for the MNIST dataset. The visualization of training progress helps assess whether the model is learning effectively and whether it might be overfitting. Overall, the code is well-structured and provides a clear understanding of your model's configuration and training process.

TASK 2:

The database structure in the example I provided for the library management system is organized into multiple tables: **authors, genres, books, and patrons**. Each table serves a specific purpose and is related to other tables in a way that represents real-world relationships and supports efficient data retrieval and management.

Here's an explanation of the organization and the reasons for choosing this structure:

Authors Table (authors):

This table stores information about authors, including their unique author_id and author_name.

Reason: Separating authors into their own table allows for efficient author management. It avoids data redundancy because multiple books can have the same author without duplicating author information.

Genres Table (genres):

This table stores information about book genres, including their unique genre_id and genre_name.

Reason: Similar to authors, genres are separated into their own table to avoid data redundancy. Multiple books can share the same genre without duplicating genre information.

Books Table (books):

This table stores information about books, including their unique book_id, title, and foreign keys author_id and genre_id.

Reason: By using foreign keys for author and genre, we establish relationships with the authors and genres tables. This ensures data consistency and referential integrity, as books can only reference authors and genres that exist in their respective tables. This structure also enables efficient querying and searching for books by author or genre.

Patrons Table (patrons):

This table stores information about library patrons, including their unique patron_id and patron_name.

Reason: In a library management system, it's essential to maintain a list of library patrons. This table allows tracking patrons' information and their interactions with books, such as checkouts and returns.

Reasons for Choosing this Structure:

Normalization: The structure follows the principles of database normalization, specifically the first three normal forms (1NF, 2NF, and 3NF), to minimize data redundancy and ensure data integrity.

Relationships: By using foreign keys, the structure establishes relationships between entities (e.g., books, authors, genres), allowing for efficient data retrieval and maintaining referential integrity.

Flexibility: This structure accommodates various relationships between books, authors, and genres. A book can have multiple authors, and authors can write multiple books. Similarly, multiple books can belong to the same genre, and a book can belong to multiple genres if needed.

Efficiency: With proper indexing and relationships, this structure enables efficient querying for tasks such as retrieving all books by a specific author or genre.

Scalability: As the library grows, this structure can scale to handle a more extensive collection of books, authors, genres, and patrons without significant changes.

Overall, the chosen structure optimally represents the relationships and data requirements of a library management system, promoting data consistency, efficiency, and maintainability. It adheres to established database design principles, ensuring that the database can effectively handle the tasks required by such a system.

TASK 3:

API Key Configuration:

I start by configuring my API key for Google Maps in the api_key variable. It's a good practice to store sensitive information like API keys in a separate configuration file or environment variables for security.

geocode_address Function:

I define a function geocode_address(address) to perform geocoding.

Inside the function, I construct the API request URL by specifying the base URL and query parameters, including the address and my API key.

I use a try...except block to handle potential exceptions during the API request.

If the response status code is 200 (indicating success), I parse the JSON response to extract the latitude and longitude coordinates from the response data.

I check the "status" field in the response to ensure the geocoding was successful ('OK'). If successful, I return the coordinates; otherwise, I print an error message.

Example Usage:

I provide an example address, "1600 Amphitheatre Parkway, Mountain View, CA," to be geocoded.

I call the geocode_address function with this address and store the result (coordinates) in the coordinates variable.

If coordinates are returned (not None), I print the latitude and longitude; otherwise, I print an error message.

Data Processing Approach:

My code uses the requests library to send an HTTP GET request to the Google Maps Geocoding API.

It processes the response data in JSON format, extracting relevant information such as latitude and longitude.

I've implemented error handling to account for potential issues with the API request or response.

Code Organization:

My code is well-structured, with clear comments explaining each part of the script.

I encapsulated the geocoding functionality within the geocode_address function, making it reusable for different addresses.

I provided a clear example of how to use the function by geocoding a specific address and printing the results.

Overall, my code is organized and follows best practices for making API requests and processing the received data. It demonstrates a straightforward and effective way to integrate with the Google Maps Geocoding API in Python.