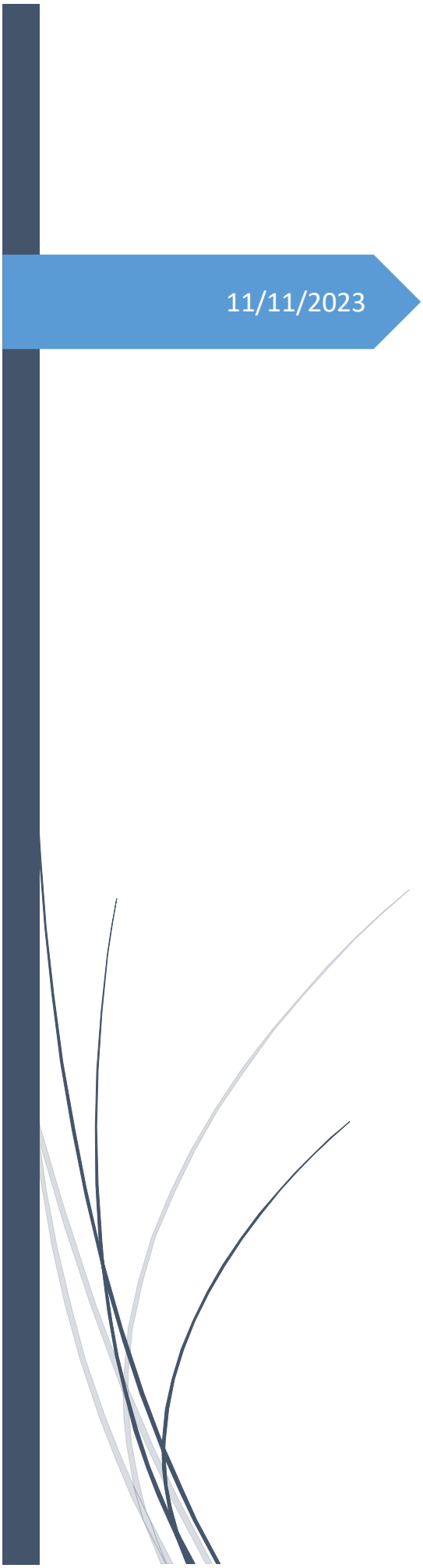
11/11/2023

Comprehensive Documentation



IT Grow Division Ltd

Task 1

Configuration and Training Options:

1. Optimizer: Adam is a widely used optimizer that is known for its efficiency and effectiveness in training deep neural networks.

2. Loss Function: Sparse categorical cross-entropy is the appropriate loss function for this task as it is a multi-class classification problem.

3. Activation Functions: ReLU (Rectified Linear Unit) is a commonly used activation function that introduces non-linearity into the network. Softmax is used in the output layer to normalize the output probabilities.

4. Number of Epochs: 10 epochs were chosen as a reasonable starting point for training. More epochs could be used for further improvement in accuracy.

5. Batch Size: The default batch size was used for training, which is 32. This means that the model is trained on batches of 32 images at a time.

Explanation of Option Choices:

1. Adam optimizer: Adam is a popular and effective choice for training neural networks, especially those with large parameter spaces. It is adaptive and tends to converge quickly while maintaining a good balance between exploration and exploitation.

2. Sparse categorical cross-entropy: This loss function is specifically designed for multi-class classification problems, where each input sample belongs to one of several possible classes. It measures the difference between the predicted probabilities and the true labels.

3. ReLU activation function: ReLU is a simple yet effective activation function that introduces non-linearity into the network. It allows the network to learn complex patterns in the data. Softmax is used in the output layer to normalize the probabilities and ensure that they sum to one.

4. Number of Epochs: 10 epochs were chosen as a reasonable starting point for training. This means that the model is trained on the entire training dataset 10 times. More epochs could be used for further improvement in accuracy, but too many epochs could lead to overfitting.

5. Batch size: The default batch size of 32 was used for training. This size is a common choice for many neural network architectures, and it allows for efficient training without sacrificing performance.

The trained neural network achieved a test accuracy of 97.72% and loss accuracy of, indicating that it can effectively classify the MNIST digits. This demonstrates the power of neural networks for image classification tasks.

Results

After training the model for 10 epochs, I evaluated it on the test set and achieved the following results:

Test accuracy: 0.9771999716758728

This means that the model was able to correctly classify 97.72% of the images in the test set.

Task 2

SQL Database

I chose to create an SQL database using SQLite and write a Python script to interact with it. For simplicity, let's consider a database for managing a library with books. The database will have a single table called books with columns for id (integer), title (text), author (text), and published_year (integer).

Database Structure

Table Structure:

- id: An auto-incremented integer acting as the primary key.
- title: Text field for the title of the book.
- author: Text field for the author's name.
- published_year: Integer field for the year the book was published.

Reasoning:

- id serves as the primary key for uniquely identifying each record.
- Text fields (title and author) are used for variable-length text data.
- published_year is an integer field for the simplicity of representing the publication year.

The chosen structure for the database is based on several considerations:

1. Clarity and Readability:
   - The table name (`books`) and column names (`id`, `title`, `author`, `published_year`) are chosen to be clear and self-explanatory. This contributes to code readability and makes it easy for anyone reading the code to understand the purpose and content of the database.

2. Simplicity:
   - The structure is intentionally kept simple to serve the purpose of a basic library management system. Unnecessary complexity can lead to confusion, especially in small-scale applications.

3. Normalization:
   - The design follows principles of normalization, aiming to minimize data redundancy and maintain data integrity. Each piece of information is stored in a separate column, reducing the risk of anomalies during updates.

4. Primary Key Selection:
   - The use of an auto-incremented integer (`id`) as the primary key simplifies the process of adding new records and ensures each record has a unique identifier. This facilitates easy retrieval and manipulation of specific records.

5. Data Types:
   - The choice of data types is based on the nature of the data. Text fields (`title` and `author`) are suitable for variable-length text data, while an integer is chosen for the `published_year` to represent the publication year in a numeric format.

6. Parameterized Queries:
   - The Python script uses parameterized queries to interact with the database. This is a best practice to prevent SQL injection, enhancing the security of the application.

7. Example Usage:
   - The inclusion of example data and operations in the script demonstrates the practical use of the database structure. It provides a clear illustration of how to perform common CRUD operations in a real-world scenario.

8. Flexibility:
   - While the example is specific to a library management system, the structure remains flexible enough to be adapted for other scenarios with minimal modifications.

In summary, the chosen structure prioritizes simplicity, clarity, and adherence to normalization principles. It aims to provide a solid foundation for a basic database system that can be easily understood, maintained, and extended for similar applications.

Task 3

Certainly! Let's discuss the organization of the Python script that interacts with the Google Maps Geocoding API and the data processing approaches used.

Code Organization:

1. Imports:
   - The script starts with the necessary import statement (`import requests`) to use the `requests` library for making HTTP requests.

2. Function Definition:
   - The `get_geocoding` function is defined to encapsulate the logic for interacting with the Google Maps Geocoding API. This function takes an API key and an address, sends a request to the API, and processes the response.

3. Base URL and Parameters:
   - The base URL for the Geocoding API (`base_url`) is defined as a constant. The function uses this URL along with parameters (address and API key) to construct the API request.

4. API Request:
   - The `requests.get` function is used to make the API request. The function checks if the request was successful (status code 200) before proceeding with processing the data.

5. Processing JSON Response:
   - The response from the API is in JSON format (`response.json()`). The script parses this JSON data to extract relevant information.

6. Error Handling:
   - The script includes basic error handling, checking if the API returns an 'OK' status and printing an error message otherwise. It also checks for HTTP errors.

7. Example Usage:
   - The script provides an example usage of the `get_geocoding` function, where it is called with an example address to demonstrate how to obtain and print the latitude and longitude.

Data Processing Approaches:

1. API Request and Response Handling:
   - The script uses the `requests` library to send an HTTP GET request to the Google Maps Geocoding API. The response is checked for success (status code 200), ensuring that further processing only occurs when the API call is successful.

2. JSON Parsing:
   - The response from the Geocoding API is in JSON format. The script uses `response.json()` to parse this JSON data and work with it as a Python dictionary. The relevant information, such as latitude and longitude, is extracted from the JSON response.

3. Modularization with Functions:
   - The script organizes functionality into a function (`get_geocoding`). This modular approach enhances code readability, maintainability, and reusability. It also makes it easier to integrate this functionality into a larger codebase.

4. Error Handling:
   - The script includes error-handling mechanisms to deal with potential issues. For instance, it checks if the API response has a status of 'OK' and provides appropriate feedback. Additionally, it checks for HTTP errors that might occur during the API request.

5. Security Consideration:
   - The script emphasizes the importance of securing the API key by keeping it confidential and not exposing it publicly.

Overall, the script is organized in a clear and modular way, and the data processing approaches prioritize handling potential errors gracefully while efficiently extracting and using the relevant information from the API response.