DOCUMENTATION

Prepared By:
Khandaker Tayef Shahriar
Email: tayef12@gmail.com

Contents

1. Task 1: Creating a neural network

1.1. Overview

This code demonstrates how to build, train, and evaluate a Convolutional Neural Network (CNN) for the task of classifying handwritten digits from the MNIST dataset. The code covers data loading, model architecture definition, training, and evaluation.

1.2. Prerequisites

- Python 3.x
- TensorFlow/Keras (for neural network modeling)
- NumPy (for data manipulation)
- Matplotlib (for visualization)

1.3. Installation

- Install Python
- Install Required Packages

1.4. Function Details

load_dataset(): Loads and preprocesses the MNIST dataset, returning training and testing data and labels.

prep_pixels(train, test): Normalizes pixel values in the dataset to the range [0, 1].

define_model(): Defines the architecture of the CNN model.

evaluate_model(dataX, dataY, n_folds=5): Performs K-Fold cross-validation to evaluate the model's performance.

1.5. Conclusion

This code demonstrates the complete pipeline for building, training, and evaluating a CNN model for MNIST digit classification. The model architecture can be changed as needed to solve various image classification tasks.

2. Task 2: Working with databases

2.1. Overview

This Django view code is designed for a student management application. It allows users to perform essential CRUD (Create, Read, Update, Delete) operations on student records in a database.

2.2. Create Student Record

- Users can input student details (name, roll number, address, mobile number) in a form.
- Upon form submission, the code validates the input data and inserts a new student record into the database.

2.3. Read Student Records

- All existing student records are retrieved from the database and displayed on the homepage.
- The code uses Django's Object-Relational Mapping (ORM) to retrieve records from the "studentapp_student" table.

2.4. Update Student Record

- Users can edit the details of an existing student record.
- The code provides a form with pre-filled data for editing.
- Upon form submission, the code updates the corresponding student record in the database.

2.5. Delete Student Record

- Users can delete a student record by clicking a delete button.
- The code handles the deletion operation and removes the record from the database.

2.6. Installation

Ensure you have a Django project set up. If not, create one using django-admin startproject projectname.

2.7. App Creation

Create an app within your project where you want to define the views: python manage.py startapp studentapp.

2.8. Model Setup

Define a model for the student records in studentapp/models.py. Ensure it matches your database schema.

2.9. URL Configuration

Add URL patterns in your project's urls.py to map to these views. Import the views using from studentapp.views import homeview, deleteview, editview.

2.10. Database Migration

Create and apply migrations for your app: python manage.py makemigrations studentapp and python manage.py migrate.

2.11. Usage

- Access the homepage to view existing student records and a form for adding new records.
- Fill out the form with student details and submit it.
- The new student record will be inserted into the database.

- On the homepage, you can see a list of all existing student records.
- Click the "Edit" button to edit a student's details.
- Click the "Delete" button to remove a student record.
- When editing a student, the form will pre-fill with their existing data.
- Upon saving the edit, the record in the database will be updated.
- When deleting a student, the corresponding record will be removed from the database.
- After successful editing or deleting, users are redirected to the homepage.

2.12. Conclusion

This documentation provides a detailed guide on the capabilities, installation, and usage of the provided Django view code for managing student records. Users should adapt the code to their specific Django project and database schema.

3. Task 3: Integration with a Google API

3.1. Overview

This Python script allows to geocode a physical address into its corresponding latitude and longitude coordinates using the Google Maps Geocoding API.

3.2. Prerequisites

Before using this script, make sure you have the following prerequisites:

- Python
- Requests Library
- Google API Key

3.3. Installation

- Clone or download this repository to your local machine.
- Navigate to the directory where the script is located.

3.4. Usage

- Open a terminal or command prompt.
- Navigate to the directory where the script is located.
- Run the script using Python.

3.5. Conclusion

This script provides a simple way to leverage the Google Maps Geocoding API for converting addresses into geographic coordinates.