DA5402 : ML Operations Lab 3 Report
Handwritten Digit Recognition System

Sahil Kokare Mm21b036

February 19, 2025

Abstract

This report details the design, implementation, and evaluation of a handwritten digit recognition system. The system utilizes a custom-built, three-layer dense neural network trained on the MNIST dataset. The system comprises a model training pipeline, a RESTful API for model deployment, and a graphical user interface (GUI) for interactive digit drawing and prediction.

1 Introduction

This report presents a technical overview of a handwritten digit recognition system. The system leverages machine learning techniques, specifically a dense neural network, to recognize digits drawn by users.

2 System Architecture

The system is structured into the following key components:

- **Model Training (model_factory.py, utils.py, dense_neural_class.py):**
  - Utilizes the MNIST dataset for training and testing.
  - Implements a custom dense neural network with ReLU and Softmax activation functions.
  - Employs batch gradient descent for model optimization.
  - Serializes the trained model using pickle for persistence.
- **API Deployment (fast_api.py):**
  - Leverages FastAPI to create a RESTful API.
  - Loads the trained model upon startup.
  - Processes image uploads, performs prediction, and returns the predicted digit.

- **GUI Application (drawing_app.py):**
  - Uses Tkinter for creating a drawing canvas.
  - Captures user-drawn digits and converts them to a 28x28 grayscale image.
  - Sends the image to the API for prediction and displays the result.
- **Supporting Utilities (utils.py):**
  - Provides essential functions for neural network operations, including activation functions, forward and backward propagation, weight updates, and data preprocessing.

3 Implementation Details

3.1 Model Training

The `model_factory.py` script loads the MNIST dataset. The `Dense_Neural_Diy` class encapsulates the neural network architecture. The `utils.py` module implements the core neural network logic. The model is trained in two phases, first with a high batch size and low epochs, then with low batch size and high epochs. The trained model is serialized using `pickle`.

3.2 API Deployment

FastAPI is used to create the RESTful API. The `fast_api.py` script loads the serialized model. The `/predict` endpoint accepts image uploads. The uploaded image is preprocessed before being fed to the model. The API returns the predicted digit in JSON format.

3.3 GUI Application

Tkinter is used to create the drawing canvas. The user can draw digits using the mouse. The drawn digit is scaled down to a 28x28 grayscale image. The image is sent to the API using `requests.post`. The predicted digit is displayed in a message box.

4 Evaluation

The model's performance was evaluated on the MNIST test set. The trained model achieved high accuracy. Qualitative evaluation was performed using the GUI application.

5 How To Use The Application

5.1 Get a copy of the repo

Start by getting yourself a copy of the project.

```
git clone https://github.com/SahilKokare-Mm21b036/assignment-03-SahilKokare-Mm21
```

And unzip the model training dataset, which was zipped and committed to the repo as the files were too large otherwise.

```
unzip mnist.zip
```

5.2 Create a Python Virtual Environment

It's recommended to create a virtual environment to isolate the project's dependencies. Open your terminal and execute the following commands:

```
python3 -m venv venv
source venv/bin/activate
```

This creates a virtual environment named `venv` and activates it.

5.3 Install Required Packages

Install the necessary Python packages using `pip`:

```
pip install -r requirements.txt
```

5.4 Start the FastAPI Server

Open a new terminal window (or tab) and activate the same virtual environment:

```
source venv/bin/activate
```

Then, launch the FastAPI server using `uvicorn`:

```
uvicorn fast_api:app --reload
```

This command starts the server and enables automatic reloading whenever you make changes to the `fast_api.py` file. The server will typically run on `http://127.0.0.1:8000`.

5.5 Run the Drawing Application

Open a new terminal window (or tab) and activate the same virtual environment:

```
source venv/bin/activate
```

Then, run the drawing application:

```
python drawing_app.py
```

The GUI window will appear, allowing you to draw digits and get predictions.

5.6 Stopping the Application

To stop the FastAPI server, press `Ctrl+C` in the terminal where it's running. To close the drawing application, simply close the GUI window. To deactivate the virtual environment, type `deactivate` in the terminal.

```
deactivate
```

6 Conclusion

This report detailed the development of a comprehensive handwritten digit recognition system, encompassing model training, API deployment, and a user-friendly graphical interface. The system successfully demonstrates the application of a custom-built dense neural network for image classification, achieving high accuracy on the MNIST dataset. By decoupling the model and UI components through a RESTful API, the system achieves a modular and scalable architecture. The Tkinter-based GUI provides an intuitive platform for users to interact with the model, showcasing the practical application of machine learning in real-time digit recognition.

The application last checked was found to be a working prototype. In case of any issues, feel free to create one on the github repository. Thank you for your time to read this report!