Alphanumeric Character Classification System Documentation

Overview

The Alphanumeric Character Classification System, is a no-framework, Kotlin-based solution for classifying alphanumeric characters in images. The system leverages the EMNIST dataset for training and evaluation and provides a REST API for model management and image classification.

Current Release

- Version: **1.0 Beta**

Key Technologies

- **Gradle**: 8.7
- **JVM**: 20
- **Kotlin**: 2.0

Dataset

The EMNIST dataset is used as the data source for training and testing the classification models. Ensure the data files are downloaded and placed in the `/data` directory.

Dataset Source: EMNIST on Kaggle

Setup Instructions

Prerequisites

1. Ensure **JAVA_HOME** is set to JDK 20.
2. Clone the repository:
3. `git clone https://github.com/ExaggeratedRumors/alphanumeric-regognizer.git`
4. Download and unzip the EMNIST data into the `/data` directory if training/testing is required.

Execution Steps

Model Training

Run the following command to train a model:

```
./gradlew trainModel --args='MODEL_NAME EPOCHS DATA_SIZE'
```

Example:

```
./gradlew trainModel --args='balanced_50e_1c_1d 50 10000'
```

Model Testing

Run the following command to test a trained model:

```
./gradlew testModel --args='MODEL_NAME DATA_SIZE'
```

Example:

```
./gradlew testModel --args='balanced_50e_1c_1d 10000'
```

Starting the Server

Run the following command to start the server:

```
./gradlew runServer --args='PORT'
```

Example:

```
./gradlew runServer --args='8080'
```

Output

- Trained models are stored in the `/models` directory.
- Each model consists of:
    - `.metadata` file containing hyperparameters.
    - `.model` file containing model weights.

REST API Endpoints

1. Model Management

`GET /models`

Retrieve a list of all trained models with their parameters.

`GET /data`

Retrieve a list of all available training and test datasets.

`GET /status/{modelName}`

Get the status of a training model.

`POST /train`

Train a new model using specified parameters and layers.

Request Example:

```
{
    "modelName": "testModel",
    "trainDataPath": "emnist-balanced-train-images-idx3-ubyte",
    "trainLabelsPath": "emnist-balanced-train-labels-idx1-ubyte",
    "trainDataSize": 1000,
    "testDataPath": "emnist-balanced-test-images-idx3-ubyte",
    "testLabelsPath": "emnist-balanced-test-labels-idx1-ubyte",
    "testDataSize": 1000,
    "epochs": 100,
    "learningRate": 0.01,
    "batchSize": 1,
    "layers": [
        {
            "type": "Input",
            "height": 28,
            "width": 28,
            "channels": 1
        },
        {
            "type": "Conv",
            "filtersAmount": 8,
            "kernel": 3,
            "activation": "relu",
            "weightRange": 0.01
        },
        {
            "type": "MaxPool",
            "poolSize": 2,
        },
        {
          "type": "Flatten"
        },
        {
            "type": "Dense",
            "neurons": 10,
            "activation": "relu",
            "weightRange": 0.01
        },
        {
            "type": "Dropout",
            "rate": 0.1
        },
        {
            "type": "Dense",
            "neurons": 47,
            "activation": "softmax"
        }
    ]
}
```

Supported Layer Types:

- Input
- Conv
- MaxPool
- Flatten
- Dense
- Dropout

`DELETE /model/{name}`

Delete a trained model by name.

2. Classification

`POST /classify`

Classify an image using a specified model.

Headers:

- `Content-Type`: `image/png`
- `Model-Name`: The name of the trained model.

Content: Binary image data.

Notes

- The server is required to run for REST API interactions.
- Ensure all data files and trained models are correctly placed in their respective directories.
- Use proper naming conventions for model files to avoid conflicts.