Image Forensics Project

Tampering Localization & Source Camera Identification

This project combines mobile and backend technology with deep learning models for a full-stack digital image forensics solution.

Components:

- Flutter frontend UI (Android/iOS/Desktop)
- Spring Boot backend API
- Tampering Localization using ConvNeXt + MMSegmentation
- Source Camera Identification (SCI) using PRNU + FFDNet + ResNet

Project Structure:

```
project-root/
 frontend/                   # Flutter app
    lib/, assets/, pubspec.yaml, etc.
 backend/                    # Spring Boot API server
    src/, pom.xml, etc.
 mmseg/                      # MMSegmentation framework (Tampering)
 ffdnet_tf/                  # FFDNet denoising in TensorFlow
 prnu_utils/                 # PRNU extraction scripts
 sci_train.py                # Train SCI classifier (ResNet)
 train.py                    # Train tampering model (ConvNeXt)
 interface.py                # (Optional) Tkinter desktop UI
 data/                       # Dataset root
    CASIA2/                 # Tampering dataset
    custom/                 # SCI dataset (D01D35)
```

Python Environment

```
$ conda create -n image-forensics python=3.8 -y
$ conda activate image-forensics
```

Dependencies

```
$ pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
$ pip install mmcv-full==1.6.2 -f https://download.openmmlab.com/mmcv/dist/cu121/torch2.0/index.html
$ pip install mmsegmentation==0.29.0
$ pip install tensorflow==2.10.0
$ pip install opencv-python scikit-image tqdm pandas matplotlib seaborn
```

Tampering Localization (ConvNeXt + MMSeg)

- Dataset: `data/CASIA2/`
- Labels: Binary masks indicating tampered regions

Train:
```
$ python train.py
```

Test:
```
$ python test.py
```

Source Camera Identification (PRNU + FFDNet + ResNet)

- Dataset: `data/custom/` with D01D35 folders
- Preprocessing: Denoising PRNU extraction Feature learning

Train:
```
$ python sci_train.py
```

Test:
```
$ python sci_test.py
```

Flutter Frontend Setup

Navigate to frontend/ directory:
```
$ flutter pub get
```

Run the app:
```
$ flutter run
```

API calls (GET/POST) are made to the Spring Boot backend.

Spring Boot Backend Setup

Navigate to backend/ directory and run:

Using Maven:
```
$ mvn clean install
$ mvn spring-boot:run
```

Or using Gradle:
```
$ ./gradlew bootRun
```

API Endpoints:
- /api/uploadImage (POST): Accepts image for tampering/SRC check
- /api/result (GET): Returns JSON result to Flutter

Integration Flow

1. Flutter UI lets user upload an image
2. Sends image to Spring Boot API via HTTP POST
3. Backend routes image to model (tampering or PRNU)
4. Result (e.g., 'Tampered', 'Camera D04') is returned as JSON
5. Flutter displays final output

All API communication uses REST and handles CORS.