Soil Erosion Prediction

Vishwa Raval

Abstract

This report describes the implementation and methodology of a machine learning framework applied to remote sensing data. The project consists of two primary scripts: 'train.py' and 'magic.py'. The 'train.py' script is responsible for handling dataset processing, model training, and evaluation, while 'magic.py' processes remote sensing imagery and extracts useful features for training.

1 Introduction

Remote sensing data provides crucial insights for various applications, including environmental monitoring and land use classification. This project focuses on training a deep learning model to analyze such data efficiently. The key goals of this project are:

- Data preprocessing for geospatial imagery
- Deep learning model implementation and training
- Evaluation of model performance

2 Methodology

The implementation is structured into two scripts:

- 'magic.py': Responsible for downloading, processing, and formatting satellite imagery.
- 'train.py': Defines the deep learning model, manages dataset loading, training, and evaluation.

2.1 Data Processing

The 'magic.py' script processes satellite images by:

1. Downloading images from an S3-compatible cloud storage.
2. Extracting specific bands (RGB) from multi-band satellite imagery.
3. Cropping images to a standard size.
4. Saving the processed images for training.

2.2 Model Training

The ‘train.py‘ script includes:

- A convolutional neural network (CNN) designed for feature extraction.
- A dataset and dataloader class for efficient training.
- An optimizer and loss function setup.
- Training and evaluation functions with logging using Weights Biases (wandb).

3 Implementation

3.1 Dataset Handling

The dataset is structured in directories, where each data sample consists of:

- Satellite image bands (stored as TIFF files)
- Corresponding labels stored in text files

The ‘EroDataset‘ class loads these images and their corresponding labels into PyTorch tensors for training.

3.2 Model Architecture

The deep learning model consists of:

- Three convolutional layers with ReLU activations and max pooling.
- A global average pooling layer to reduce dimensionality.

- A fully connected layer for prediction.

3.3 Training Pipeline

The training process involves:

1. Splitting the dataset into training, validation, and test sets.
2. Training the model using the Adam optimizer and MSE loss.
3. Evaluating the model at regular intervals.
4. Logging results using wandb.
5. Saving model checkpoints for future use.

4 Results and Conclusion

The model successfully trains on the dataset, achieving meaningful predictions. The use of wandb for tracking experiments provides insights into model performance, enabling fine-tuning of hyperparameters. Future improvements could include using additional spectral bands and implementing advanced architectures.

5 Appendix

5.1 Sample Code Snippets

5.1.1 Dataset Class

```
class EroDataset(Dataset):
    def __init__(self, data_path, split_path):
        self.folder_path_list = [Path(line) for line in
            FSToolKit.read_from_file(split_path, parent=data_path)]
        self.outputs = self.get_outputs()

    def __getitem__(self, index):
                                data = self.read_data([self.folder_path_list
```

[index]]) output = torch.Tensor([self.outputs[index]]) return (data, output)

5.1.2 Model Architecture

```
class Model(nn.Module):
    def __init__(self): super().__init__() self.conv1 = nn.Conv2d(3, 64,
        kernel_size=3, stride=1, padding=1)
        self.relu1 = nn.ReLU(inplace=True) self.maxpool1 =
        nn.MaxPool2d(kernel_size=2, stride=2) self.fc = nn.Linear(256, 1)

    def forward(self, x): x = self.conv1(x)
        x = self.relu1(x) x =
        self.maxpool1(x) return
        self.fc(x)
```