IBM DATA SCIENCE PROJECT

Land Type Classification using Sentinel-2 Satellite Images

Digital Egypt Pioneers – AI & Data Science Track

Team members:

- **Omar Masoud**
- **Omar Ali Ibrahim**
- **Rewan Mohamed**
- **Mohamed Ragab**
- **Abdelrahman Awad**
- **Youssef Hesham**

Executive Summary

This project implements a comprehensive land type classification system using deep learning to analyze Sentinel-2 satellite imagery. The system successfully classifies satellite images into ten distinct land cover categories with approximately 94% accuracy. A pre-trained ResNet-50 model was fine-tuned on the EuroSAT dataset and deployed as a web application with a user-friendly interface built using Flask and Tailwind CSS. The model provides fast, accurate classification of uploaded satellite imagery, making it a valuable tool for urban planning, environmental monitoring, and resource management.

The dataset that we used:

https://www.kaggle.com/datasets/apollo2506/eurosat-dataset

GITHUB Repo:

https://github.com/OmarAli141/Land_Type_Classification

---

1. Introduction

1.1 Background

Satellite imagery provides valuable data for monitoring Earth's surface, but manually classifying land types is time-consuming and labor-intensive. Automated classification using deep learning offers a scalable solution to this challenge.

1.2 Project Objectives

- Develop a deep neural network to classify land types from Sentinel-2 satellite imagery
- Create an intuitive web interface for uploading and analyzing satellite images
- Achieve high classification accuracy across diverse land cover types
- Deploy the solution as a scalable web application

1.3 Target Land Types

The system classifies images into ten distinct categories:

- Annual Crop
- Forest
- Herbaceous Vegetation
- Highway
- Industrial
- Pasture
- Permanent Crop
- Residential
- River
- Sea/Lake

---

2. Methodology

2.1 Data Collection and Preprocessing

Dataset: The EuroSAT dataset was utilized, containing 27,000 labeled Sentinel-2 satellite images across 10 land cover classes. Each image is a 64×64 pixel RGB representation of a specific land type.

Preprocessing Pipeline:

- Image resizing to 64×64 pixels
- Normalization using mean=[0.5, 0.5, 0.5] and std=[0.5, 0.5, 0.5]
- Data augmentation techniques applied to the training set:
    - Random horizontal and vertical flips
    - Random rotation (up to 30 degrees)
    - Color jitter (brightness, contrast, saturation, hue)

Data Splitting:

- Training set: 70% of data
- Validation set: 10% of data
- Test set: 20% of data

2.2 Data Exploration and Analysis

Exploratory Data Analysis:

- Class distribution analysis showed a balanced representation across all ten land types
- RGB channel analysis identified distinctive spectral signatures for different land types
- PCA visualization demonstrated clear separation between most land cover classes
- Identified no corrupted files or missing data in the dataset

2.3 Model Architecture

Base Model: ResNet-50 pre-trained on ImageNet

Model Adaptation:

- Replaced the final fully connected layer to output 10 classes
- Maintained the convolutional base for feature extraction
- Total parameters: 23.5 million

Training Parameters:

- Loss function: Cross-Entropy Loss
- Optimizer: Stochastic Gradient Descent (SGD)
- Learning rate: 1e-3
- Batch size: 32
- Training epochs: 10

---

3. Implementation

3.1 Model Development and Training

Training Process:

- The model was trained for 10 epochs with early stopping based on validation loss
- Best model weights were saved for deployment
- Training conducted using PyTorch on GPU acceleration when available

Evaluation Metrics:

- Precision: 0.9375
- Recall: 0.9356
- F1 Score: 0.9355

3.2 Backend Implementation

The backend was developed using **Flask**, a lightweight Python web framework, with the following core components:

1. Model Initialization

- **ResNet-50 Architecture**

2. Image Preprocessing Pipeline

- **Transformations**:
    1. Resize images to **64x64 pixels** (matching training dimensions).
    2. Convert to PyTorch tensor.
    3. Normalize pixel values with mean=[0.5, 0.5, 0.5] and std=[0.5, 0.5, 0.5] (same as training).

3. Prediction Logic

- **Workflow**:
    1. Convert uploaded image to **RGB format** (ensures 3-channel input).
    2. Apply transformations and add a batch dimension.
    3. Run inference with torch.no_grad() for efficiency.

4. API Endpoints

- **Routes**:
    - **/**: Renders the index.html template.
    - **/predict**: Handles image uploads, runs inference, and returns JSON results

5. Error Handling & Optimization

- **Key Features:**
    - **Model File Validation: Checks if ResNet-50_model.pth exists before loading.**
    - **GPU/CPU Agnostic: Forces inference on CPU via map_location=torch.device('cpu').**
    - **Temporary File Cleanup: Deletes uploaded images after prediction to avoid storage bloat.**

3.3 Frontend Interface

The frontend was built using HTML and Tailwind CSS with the following features:

- Responsive design with satellite imagery background

- Drag-and-drop file upload functionality
- Image preview before submission
- Real-time prediction results with confidence scores
- Loading animation during processing

---

4. Results and Performance Analysis

4.1 Model Performance

The trained ResNet-50 model achieved excellent performance metrics:

Metric	**Training Set**	**Test Set**
Precision	0.9375	0.9375
Recall	0.9356	0.9356
F1 Score	0.9355	0.9355

The similar performance between training and test sets indicates that the model generalizes well without overfitting.

4.2 Confusion Matrix Analysis

The confusion matrix revealed:

- Highest accuracy for water bodies (Sea/Lake and River classes)
- Some confusion between Annual Crop and Permanent Crop classes
- Minor misclassifications between Residential and Industrial areas

4.3 Precision-Recall Analysis

Precision-recall curves showed:

- All classes achieved area-under-curve values above 0.9
- Forest and Sea/Lake classes had the highest precision-recall performance
- Highway class had the lowest performance but still achieved good results

5. Deployment

5.1 System Architecture

The system follows a three-tier architecture:

1. **Data Pipeline:**
   - Preprocessing of satellite imagery
   - Feature extraction
   - Data transformation
2. **Model Layer:**
   - Pre-trained ResNet-50
   - Custom classification head
   - Inference engine
3. **Application Layer:**
   - Flask backend
   - HTML/CSS frontend
   - RESTful API endpoints

5.2 Deployment Setup

The application is deployed using:

- Flask web server for backend processing
- Static file serving for frontend assets
- Local model loading with CPU inference
- In-memory processing of uploaded images

6. Future Improvements

6.1 Model Enhancements

- Implement ensemble methods to improve classification accuracy
- Explore more advanced architectures like EfficientNet or Vision Transformer
- Add semantic segmentation capabilities for pixel-level classification

6.2 System Improvements

- Implement model quantization to reduce inference time
- Develop a RESTful API for programmatic access
- Add batch processing capabilities for multiple images
- Incorporate geospatial metadata analysis

6.3 Interface Extensions

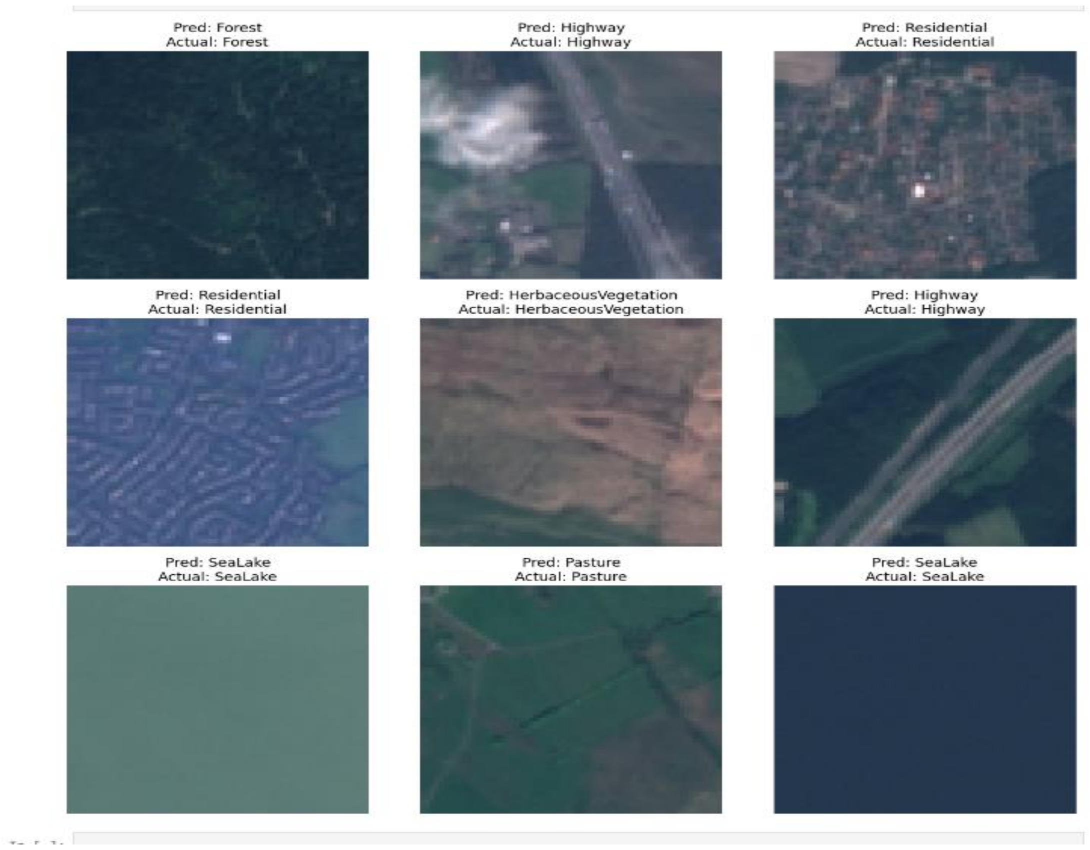
- Add visualization of class activation maps
- Implement comparison between multiple images
- Develop a dashboard for historical analysis
- Add export functionality for classification results

7. Conclusion

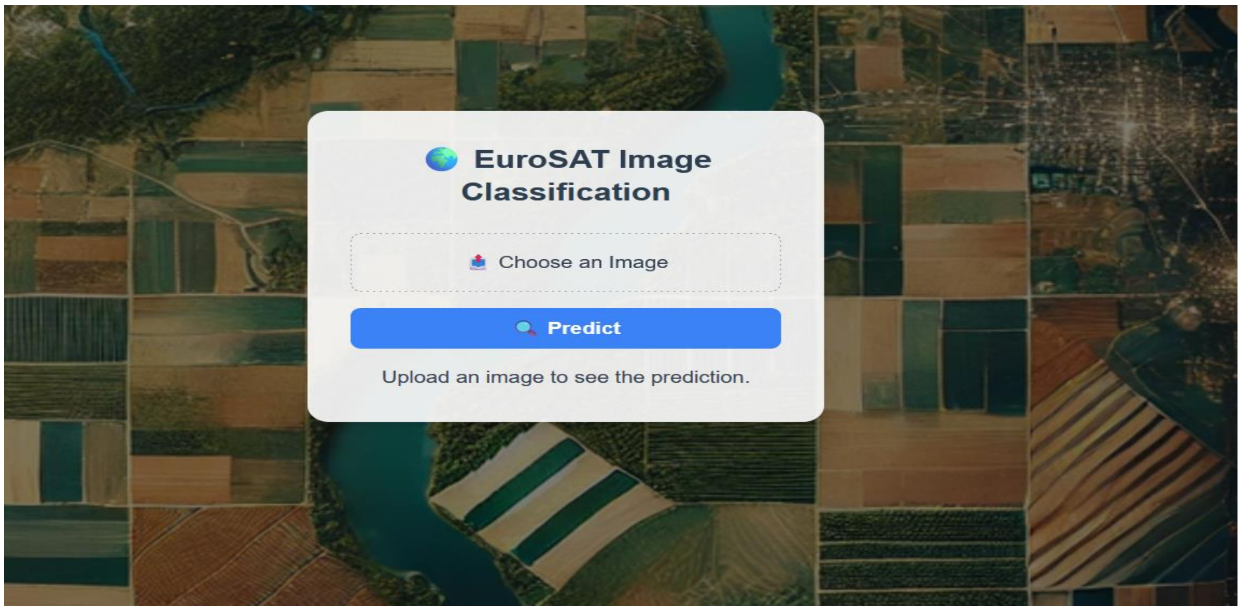
The Land Type Classification system successfully demonstrates the application of deep learning for satellite imagery analysis. By leveraging transfer learning with ResNet-50 on the EuroSAT dataset, the system achieves high accuracy in distinguishing between different land cover types. The web-based deployment provides an accessible interface for users to analyze satellite imagery without specialized GIS knowledge.

This project showcases the potential of deep learning in environmental monitoring and resource management, providing a foundation for more advanced Earth observation applications. The system's modular architecture allows for future extensions and improvements to address more complex satellite imagery analysis needs.

Screenshot of Predicting:



Screenshot of Deployments:


🌍 EuroSAT Image Classification
Choose an Image
Predict
Upload an image to see the prediction.