**Project Documentation: Land Classification Using CNN**

**1. Introduction**

**1.1 Project Overview**

This project involves developing a deep learning-based web application that classifies satellite images into two categories: **Constructed Building** or **Empty Land**. The classification model is built using **Convolutional Neural Networks (CNNs)** and trained on the **EuroSat open-source dataset**. The project also includes a Flask-based web application that allows users to upload an image for prediction.

**1.2 Objectives**

* To develop a CNN-based image classification model for land use classification.
* To create a user-friendly web application for easy image classification.
* To explore the usefulness of empty land by suggesting possible applications.
* To integrate deep learning with real-world geospatial applications.

**1.3 Applications**

* **Urban Planning**: Identifying areas for infrastructure development.
* **Real Estate**: Assessing land suitability for housing projects.
* **Agriculture**: Evaluating land for farming and forestry.
* **Environmental Studies**: Monitoring land use changes over time.

**2. Dataset Description**

**2.1 EuroSat Dataset**

* The **EuroSat dataset** consists of satellite images collected from the Sentinel-2 satellite.
* It contains **10 different classes**, representing different land use types.
* Each image is of **64x64 pixels** in size.

**2.2 Classes in the Dataset**

* **AnnualCrop** (used for empty land classification)
* **Forest**
* **HerbaceousVegetation**
* **Highway**
* **Industrial**
* **Pasture**
* **PermanentCrop**
* **Residential** (used for constructed buildings classification)
* **River**
* **SeaLake**

**2.3 Data Preprocessing**

* **Resizing** images to 64x64 pixels.
* **Normalization** using ImageNet mean and standard deviation.
* **Splitting** into training and validation sets.

**3. Model Architecture**

**3.1 Convolutional Neural Network (CNN)**

CNNs are a type of deep learning model designed for image processing. The key components of a CNN are:

**3.1.1 Convolutional Layers**

* **Feature extraction** through **filters (kernels)** that detect patterns.
* The filters slide over the image to create **feature maps**.

**3.1.2 Activation Functions**

* **ReLU (Rectified Linear Unit)** is used to introduce non-linearity into the model.

**3.1.3 Pooling Layers**

* **Max Pooling** is used to **downsample** feature maps, reducing computational cost.

**3.1.4 Fully Connected Layers**

* The extracted features are passed through **dense layers** to make predictions.

**3.1.5 Softmax Layer**

* Converts outputs into probability distributions for classification.

**3.2 Model Implementation**

**3.2.1 Network Structure**

| **Layer** | **Type** | **Filters/Neurons** | **Activation** |
| --- | --- | --- | --- |
| 1 | Conv2D | 16 | ReLU |
| 2 | MaxPooling | - | - |
| 3 | Conv2D | 32 | ReLU |
| 4 | MaxPooling | - | - |
| 5 | Conv2D | 64 | ReLU |
| 6 | MaxPooling | - | - |
| 7 | Fully Connected | 128 | ReLU |
| 8 | Output | 2 | Softmax |

**3.3 Loss Function and Optimizer**

* **Loss Function**: CrossEntropyLoss (for multi-class classification)
* **Optimizer**: Adam Optimizer (adaptive learning rate optimization)

**4. Model Training and Evaluation**

**4.1 Training Process**

* **Epochs**: 10
* **Batch Size**: 32
* **Learning Rate**: 0.001
* **Backpropagation** is used to update weights.

**4.2 Model Evaluation**

* **Validation Accuracy**: Checked on unseen data.
* **Confusion Matrix**: Evaluates model performance.
* **Precision & Recall**: Measures classification correctness.

**5. Web Application Development**

**5.1 Flask Application**

A web-based interface is developed using Flask to allow users to upload images and receive predictions.

**5.1.1 Features**

* Upload an image via web browser.
* Get a prediction (Constructed Building / Empty Land).
* If empty land, display **profitable land-use suggestions**.

**5.1.2 Flask Application Structure**

| **File** | **Description** |
| --- | --- |
| app.py | Handles backend processing and model inference |
| index.html | User interface for image upload and results |

**6. Deployment and Execution**

**6.1 Steps to Run the Project**

1. **Install dependencies**:
2. pip install torch torchvision flask pillow
3. **Run Flask application**:
4. python app.py
5. **Access Web Application**:
   * Open http://127.0.0.1:5000/

**7. Example Predictions**

**7.1 Sample Input & Output**

| **Input Image** | **Prediction** | **Suggestions (If Empty Land)** |
| --- | --- | --- |
| ![Empty Land] | Empty Land | Agriculture, Solar Panels, Real Estate |
| ![Constructed] | Constructed Building | - |

**8. Future Enhancements**

* Improve accuracy using **Transfer Learning** (ResNet, EfficientNet)
* Add more land categories for classification
* Deploy to **Cloud (AWS/GCP)** or create a **Mobile App**

**9. Conclusion**

This project successfully integrates **deep learning and Flask** for land classification, making it useful for **urban planning, real estate, and agriculture**. The model is accurate and provides meaningful insights into land usage.

**10. References**

1. **EuroSat Dataset**: <https://github.com/phelber/EuroSAT>
2. **PyTorch Documentation**: <https://pytorch.org/docs/stable/index.html>
3. **Flask Documentation**: <https://flask.palletsprojects.com/>

*This document provides a comprehensive explanation of the project, covering technical details, implementation, and future scope.*