**Mini Project: Image Classification Using CNN with OpenCV and TensorFlow**

**Project Overview:**

The goal of this mini-project is to design and implement an image classification model using Convolutional Neural Networks (CNNs). Students will learn to handle image data, preprocess it, train a CNN, improve its performance, and deploy the trained model on an open-source platform such as Hugging Face or Streamlit.

**Dataset:**

* **Dataset**: Use the Intel Image Classification dataset available on Kaggle. This dataset contains images of natural scenes categorized into six classes: 'buildings', 'forest', 'glacier', 'mountain', 'sea', and 'street'.
* **Source:** <https://www.kaggle.com/datasets/puneet6060/intel-image-classification>

**Learning Objectives:**

* Convert image data to numerical arrays using Python libraries (OpenCV).
* Perform data preprocessing and augmentation.
* Train a CNN model using TensorFlow.
* Improve model performance with various optimization techniques.
* Deploy the trained model to an open-source platform.

**Project Tasks:**

**1. Setting Up the Environment:**

* Install and import necessary Python libraries: TensorFlow, OpenCV, NumPy, Matplotlib, etc.
* Ensure access to a Jupyter Notebook or any Python IDE (e.g., VSCode, PyCharm).

**2. Data Collection and Loading:**

* Download and unzip the Intel Image Classification dataset.
* Load image data using OpenCV and convert it to arrays.
* Split the dataset into training, validation, and test sets.

**3. Data Preprocessing:**

* Resize images to a uniform size (e.g., 150x150 pixels).
* Normalize pixel values to a range of 0 to 1.
* Implement data augmentation techniques (e.g., rotation, flipping, zooming) using TensorFlow or OpenCV.

**4. Building the CNN Model:**

* Design a simple CNN architecture using TensorFlow/Keras.
  + Input layer with image dimensions (150x150x3).
  + Convolutional layers with ReLU activation and max pooling.
  + Flatten layer followed by dense layers.
  + Output layer with a softmax activation for multi-class classification.
* Compile the model using an appropriate loss function and optimizer (e.g., categorical cross-entropy, Adam optimizer).

**5. Model Training and Evaluation:**

* Train the model on the training set with validation checks.
* Plot training and validation accuracy/loss using Matplotlib.
* Evaluate the model's performance on the test set and compute metrics (e.g., accuracy, precision, recall).

**6. Model Optimization:**

* Implement techniques such as dropout, batch normalization, and learning rate scheduling to enhance performance.
* Fine-tune hyperparameters to improve accuracy.

**7. Model Deployment:**

* Save the trained model using TensorFlow's model.save() function.
* Deploy the model using a simple web app framework:
  + **Option 1**: Use Streamlit to create an interactive web app for image upload and prediction.
  + **Option 2**: Deploy the model on Hugging Face Spaces.

**Key Deliverables:**

* Python code for loading and preprocessing image data.
* CNN model architecture and training script.
* Plots of training/validation metrics.
* Deployment code for a web app interface.
* A brief report summarizing the approach, challenges faced, and improvements made.