System Overview

Existing System

Traditional skin cancer screening relies on in-person consultations with dermatologists who examine lesions visually and recommend biopsies for confirmation. Some earlier automated systems used static rule-based models or basic image processing techniques to analyze lesions. These systems were neither scalable nor accurate across different skin tones and lesion types. They also lacked adaptability and continuous learning, limiting their utility in real-world clinical scenarios. As a result, the early models had high false positive and negative rates, which could lead to misdiagnosis.

Proposed System

The proposed system is a comprehensive AI-powered web application for automated skin cancer detection. It enables users to upload dermatoscopic images which are analyzed using an ensemble of deep learning models. These models—EfficientNetV2S, EfficientNetV2M, InceptionResNetV2, and XceptionNet—are integrated into a single architecture using an averaging ensemble method. This improves accuracy and reduces bias. A simple Gradio-based interface ensures ease of use for both clinicians and laypeople. The system processes input in real time and returns a diagnosis along with a brief description of the detected condition. This architecture supports continuous improvement by retraining with new datasets.

Requirement Analysis

Software Requirements:

- Python 3.x
- TensorFlow/Keras for deep learning
- NumPy and Pandas for data handling
- Gradio for the web interface
- Google Colab or local GPU setup for model training

Hardware Requirements:

- GPU-enabled machine (e.g., NVIDIA Tesla T4 or RTX 3080)
- Minimum 16 GB RAM
- 200 GB storage for dataset and model weights

Prerequisites

Basic understanding of Python programming, image classification, and neural networks is required. Familiarity with CNNs and libraries like TensorFlow helps in understanding the implementation. A general knowledge of dermatology is also helpful to comprehend lesion types.

Development Tools Used

- Google Colab: For model training and experimentation.
- Jupyter Notebook: For local testing and debugging.
- VS Code: For editing Python scripts and web integration code.
- GitHub: For version control and collaboration.

Core Libraries and Frameworks

- TensorFlow/Keras: Model creation, training, and inference.
- Gradio: Web interface to interact with the trained model.
- Matplotlib/Seaborn: For visualizing training metrics.
- Scikit-learn: For evaluation metrics like confusion matrix, accuracy, and classification report.

Technologies Used

- Deep Learning (CNNs): For automatic feature extraction and classification.
- Ensemble Learning: To combine predictions from multiple models.
- Image Processing: For data augmentation and preprocessing.
- Web-based Inference: Real-time predictions using a browser interface.