Skin Disease Detetction using AI at the Edge

#### ABSTRACT

The **Skin Disease Detection Using AI at the Edge** project leverages artificial intelligence (AI) and edge computing to provide real-time, accurate diagnosis of various skin conditions. Using advanced computer vision techniques, a Convolutional Neural Network (CNN) model is trained on a diverse dataset of skin images to classify diseases like acne, eczema, melanoma, and psoriasis. By deploying this model on edge devices such as smartphones, wearables, or IoT sensors, the system enables instant skin disease detection without requiring continuous cloud connectivity. The lightweight AI model is optimized for edge devices through techniques like quantization and pruning, ensuring efficiency in both performance and power consumption. The edge-based approach enhances privacy, reduces latency, and ensures that users can receive immediate feedback without relying on external servers. This solution offers a cost-effective, secure, and scalable method for widespread skin disease detection, particularly in remote or underserved regions, bridging the gap between healthcare and technology.

**INDEX**

**Sr. No. Contents Page No.**

1. INTRODUCTION 01
   1. Existing System 01
   2. Proposed System 03
2. LITERATURE SURVEY 05
3. SOFTWARE REQUIREMENTS AND SPECIFICATION 07

[4. TECHNOLOGY 13](#_Toc48122)

[5. IMPLEMENTATION 18](#_Toc48123)

[6. SNAPSHOTS 29](#_Toc48124)

[7. SYSTEM TESTING 32](#_Toc48125)

[8. CONCLUSION 39](#_Toc48126)

[REFERENCES 40](#_Toc48127)

#### 

#### CHAPTER 1

#### INTRODUCTION

Skin diseases are among the most common health issues worldwide, affecting millions of people. While many of these conditions are treatable, early diagnosis is essential for effective treatment and management. However, access to dermatologists and timely consultations may not always be available, especially in remote or underserved areas. In this context, artificial intelligence (AI) offers a promising solution by automating the detection and classification of skin diseases.

The **Skin Disease Detection Using AI at the Edge** project aims to develop an AI-powered system that enables real-time detection of skin conditions directly on edge devices like smartphones, wearable devices, and IoT sensors. This approach eliminates the need for cloud-based processing, making the system more efficient, cost-effective, and secure. The core of the project is a Convolutional Neural Network (CNN) model trained on a diverse set of skin images to classify various conditions, from common issues like acne and eczema to more serious diseases such as melanoma.

Edge computing plays a pivotal role in this project by ensuring that the AI model operates locally on devices, allowing for faster predictions, improved privacy, and reduced reliance on continuous internet connectivity. This also makes the solution accessible in regions where internet access may be limited or unstable.

By combining AI with edge computing, the proposed system offers a scalable, user-friendly, and efficient solution for skin disease detection. This approach empowers individuals to monitor and diagnose their skin health independently, offering a more affordable and accessible alternative to traditional healthcare services.

**1.1 EXISTING SYSTEM:**

In the existing landscape, skin disease detection is primarily carried out through traditional methods, relying on expert dermatologists for visual examination and biopsy-based diagnoses. However, advancements in AI and machine learning (ML) have led to the development of digital tools aimed at automating skin disease detection. Below is an overview of existing systems and their limitations:

1. **Manual Diagnosis by Dermatologists**: Traditional skin disease diagnosis involves physical examination by dermatologists, who visually inspect the patient's skin condition and, in some cases, conduct biopsies for confirmation. This method, while effective, is time-consuming, expensive, and dependent on the availability of skilled professionals. Additionally, access to dermatologists may be limited in rural or underserved regions, leading to delayed or missed diagnoses.
2. **AI-Based Cloud Solutions**: AI-based systems, which use deep learning algorithms like Convolutional Neural Networks (CNNs), have been developed to assist dermatologists in diagnosing skin conditions. These systems analyze skin images to classify diseases like melanoma, acne, psoriasis, and eczema. Notable cloud-based platforms like **SkinVision** and **MoleScope** provide users with an easy-to-use interface where they can upload skin images for analysis. These systems often rely on pretrained models, which are continuously updated using cloud infrastructure. However, these cloud-based systems require an internet connection to function, which may be a limitation in rural areas with poor connectivity. Furthermore, sending sensitive medical data to the cloud raises concerns about privacy and security.
3. **Mobile Applications for Skin Disease Detection**: Several mobile applications have been developed to offer skin disease detection using AI. For example, apps like **Dermatology AI** and **Skin App** allow users to take photos of their skin conditions and receive an immediate diagnosis or recommendation. These apps generally rely on AI models trained on large datasets, and the results are delivered to users in real-time. However, many of these mobile apps still require an active internet connection to upload images to a cloud server for analysis. The reliance on cloud infrastructure can create latency issues and raise concerns about user privacy.
4. **Edge AI in Healthcare**: Some systems have begun to integrate edge computing into healthcare, enabling AI-based processing to take place on local devices like smartphones, wearables, and IoT devices. These systems aim to provide real-time, privacy-conscious, and efficient analysis of medical data without needing constant internet access. However, current edge-based solutions for skin disease detection are still limited in scope. While some research and development have been done to optimize AI models for edge devices, the deployment of sophisticated, deep-learning models in mobile or wearable devices for accurate skin disease detection remains a challenge. Optimizing models for real-time processing on edge devices without compromising accuracy or performance is still a work in progress.
5. **Telemedicine Integration**: Some healthcare platforms are integrating AI-powered skin disease detection into telemedicine solutions. For instance, **Tele dermatology** platforms enable patients to upload images of their skin conditions, which are then analysed by AI models. These solutions are generally connected to dermatologists, who confirm the diagnosis and suggest treatments. Although this improves access to dermatology services, it still requires cloud-based processing and is dependent on the availability of dermatologists.

**1.2 PROPOSED SYSTEM:**

The proposed system, **Skin Disease Detection Using AI at the Edge**, aims to revolutionize the current methods of skin disease diagnosis by utilizing artificial intelligence (AI) in combination with edge computing. Unlike existing systems that rely heavily on cloud-based processing, our solution focuses on providing real-time skin disease detection on edge devices like smartphones, wearables, and IoT sensors, ensuring privacy, faster response times, and reduced reliance on internet connectivity.

**Key Features:**

1. **AI-Powered Disease Detection**: The core of the system is an AI model, specifically a Convolutional Neural Network (CNN), which is trained to classify various skin conditions from images. The CNN model will be capable of identifying common skin conditions such as acne, eczema, psoriasis, and melanoma, among others. The model is trained using a large dataset of labeled skin images, ensuring its ability to detect a wide range of skin diseases with high accuracy.
2. **Edge Computing Integration**: The primary innovation of the proposed system is the use of edge computing for real-time skin disease detection. Instead of relying on cloud servers for image processing, the trained AI model will be deployed on edge devices, allowing for local processing and instant results. By performing the computations on the device, the system reduces latency and minimizes the need for continuous internet connectivity. This ensures that users can receive immediate feedback on their skin condition, regardless of their network connectivity.
3. **Optimized AI Model for Edge Devices**: AI models are often too large and computationally intensive for resource-constrained devices. To address this, the model will be optimized for edge computing using techniques like model pruning, quantization, and knowledge distillation. These optimization methods reduce the model size and computation load, making it feasible to run the AI model on smartphones and wearables with limited processing power while maintaining a high level of accuracy.
4. **User-Friendly Interface**: The system will feature a simple and intuitive user interface (UI) that allows users to capture clear images of their skin using the device's camera. The interface will guide users to ensure that the images are of sufficient quality for analysis. Once the image is captured, the system will process it locally and provide an immediate diagnosis, along with recommendations for further action, such as seeking medical attention or applying specific treatments.
5. **Privacy and Security**: A key advantage of the proposed system is that all data processing occurs locally on the user's device. This approach ensures that sensitive data, such as images of the user's skin, never leave the device, providing enhanced privacy and security compared to cloud-based systems. The system will comply with healthcare data regulations like HIPAA (Health Insurance Portability and Accountability Act), ensuring that user data is handled securely and responsibly

**CHAPTER 2**

**LITERATURE SURVEY**

1. **AI in Dermatology**: Several studies have highlighted the potential of artificial intelligence (AI) in dermatology, particularly in the detection and classification of skin diseases. Convolutional Neural Networks (CNNs) have emerged as the most effective deep learning models for image-based disease diagnosis. In 2016, a study by **Esteva et al.** demonstrated that deep learning algorithms could match or even outperform dermatologists in classifying skin cancer, specifically melanoma, from images. Their research provided evidence of the ability of AI to analyze dermatological images at a high level of accuracy, providing valuable insights into the potential of AI in dermatology (Esteva, A., Kuprel, B., Novoa, R. A., et al., 2016, *Nature*).
2. **Edge Computing for Healthcare**: Edge computing, the practice of processing data locally on devices instead of transmitting it to central servers, has gained significant traction in healthcare applications due to its advantages in real-time processing, security, and bandwidth usage. A paper by **Zhang et al.** (2020) discusses how edge computing can revolutionize healthcare by enabling immediate data analysis on devices like smartphones and wearables. This approach minimizes delays and improves privacy since sensitive data is processed locally. The research emphasizes the potential of combining AI with edge computing for applications like health monitoring, diagnostics, and patient care.
3. **Skin Disease Classification with CNN**: CNNs have shown outstanding performance in image classification tasks, including skin disease detection. A study by **Liu et al.** (2017) demonstrated that CNN-based models could successfully classify a variety of dermatological diseases, including benign and malignant conditions. By using large-scale datasets such as the ISIC (International Skin Imaging Collaboration) archive, their model achieved high accuracy and robustness, highlighting the potential of AI in skin disease diagnosis (Liu, Y., et al., 2017, *IEEE Transactions on Biomedical Engineering*). Furthermore, **Cai et al.** (2020) explored using transfer learning to fine-tune pre-trained models for skin disease detection, improving model performance and reducing the time required for training on smaller datasets.
4. **Mobile AI for Dermatology**: The development of AI models specifically for mobile devices is critical for the success of real-time skin disease detection. Researchers like

**Gonzalez et al.** (2018) have explored mobile-based platforms for skin condition diagnosis.

Their work demonstrated the feasibility of integrating AI models into mobile applications, allowing users to take photos of their skin and receive instant feedback on potential conditions. The research also emphasized the need for user-friendly interfaces to ensure the accessibility of these applications for both healthcare professionals and the general public.

1. **Model Optimization for Edge Devices**: AI models are often too large to run efficiently on edge devices, necessitating model optimization techniques. Studies on model pruning, quantization, and distillation (e.g., **Han et al.**, 2016) have proven effective in reducing the size and computational demands of deep learning models without significantly sacrificing accuracy. In the context of dermatology, optimizing AI models for deployment on smartphones or wearable devices enables real-time skin disease analysis, enhancing the accessibility and usability of these systems in resource-constrained environments.

**CHAPTER 3**

**SOFTWARE REQUIREMENTS**

#### SPECIFICATION

#### HARDWARE REQUIREMENTS

* Smartphone (Android/iOS) with camera for capturing images of the skin.
* Edge device (e.g., Raspberry Pi, NVIDIA Jetson, or Coral Edge TPU) for local processing of AI models.
* Sufficient storage space (at least 4GB of free storage) for app and model data.
* High-speed internet connection for model updates and cloud integration (if applicable).
* Graphics Processing Unit (GPU) for training the AI model (optional but recommended).
* Sufficient RAM (at least 2GB for mobile and 4GB+ for edge devices) for running the app and AI model.

**SOFTWARE REQUIREMENTS**

* Python 3.x for backend model development and AI algorithms.
* TensorFlow, Keras, or PyTorch for building and training AI models.
* OpenCV for image processing and analysis.
* TensorFlow Lite or ONNX for optimizing models for edge devices.
* Flutter or React Native for mobile app development (cross-platform).
* Android Studio or Xcode for building and testing the mobile app.
* Firebase or similar services for cloud integration (optional).
* Visual Studio Code or PyCharm for development and debugging.

The Software Requirements Specification (SRS) for the **Skin Disease Detection Using AI at the Edge** system defines the functional and non-functional requirements that the system should meet. This document outlines the system's intended functionality, its design constraints, and the standards that the system must adhere to. Below are the key software requirements for the project:

1. **Functional Requirements:**

User Authentication**:** The system should allow users to register and log in securely to the mobile application using email, social media accounts, or biometric authentication.

* + **Image Capture**: The application must enable users to capture images of skin conditions using the device’s camera, with real-time feedback on the captured image quality (e.g., clarity, focus).
  + **Skin Disease Detection**: The AI model will process the captured image and provide a prediction regarding potential skin diseases (e.g., acne, eczema, melanoma), with a confidence score.
  + **Real-time Results**: The AI model should generate results in real-time on the edge device without the need for cloud processing.
  + **User Interface**: The system should provide an intuitive and easy-to-navigate interface for users to view the diagnosis, associated recommendations, and treatment information.
  + **History and Reports**: Users should be able to track their previous diagnoses, view historical records, and download detailed reports for consultation with healthcare professionals.
  + **Model Updates**: The system should allow easy over-the-air (OTA) updates of the AI model, improving accuracy and performance over time.
  + **Data Storage**: User data and model information should be securely stored on the mobile device (or optionally in the cloud if required) in compliance with data privacy regulations.
  + **Offline Functionality**: The system should function offline, with the AI model capable of detecting skin diseases without requiring continuous internet connectivity.

1. **Non-Functional Requirements:** 
   * **Performance**: The system must provide real-time skin disease detection with minimal delay (within 5 seconds) after image capture.
   * **Scalability**: The system should be scalable to accommodate a growing user base and include the ability to add new diseases to the detection model.
   * **Reliability**: The application must be robust and reliable, capable of handling errors gracefully without crashing, even under heavy use.
   * **Security**: All user data should be encrypted and stored securely on the device. The system must adhere to industry standards for data security and user privacy (e.g., HIPAA compliance for healthcare data).
   * **Usability**: The application should be user-friendly, with a simple interface that users with no technical expertise can easily understand and use.
   * **Compatibility**: The mobile application should be compatible with both Android and iOS devices. The edge device should support models built using TensorFlow Lite or

ONNX.

* + **Maintainability**: The system should be easy to maintain, with clear code documentation and a modular design that allows easy updates and improvements.
  + **Localization and Accessibility**: The system should support multiple languages and be accessible to people with disabilities (e.g., voice commands for visually impaired users).

1. **System Requirements:** 
   * **Operating System**:
     + Mobile: Android 9.0+ or iOS 12+.
     + Edge Device: Linux-based OS (e.g., Raspbian for Raspberry Pi).
   * **Programming Languages**:
     + Python for backend AI model development.
     + Dart (Flutter) or JavaScript (React Native) for mobile app development.
   * **Libraries and Frameworks**:
     + TensorFlow, Keras, or PyTorch for AI model development. o TensorFlow Lite for edge device optimization. o OpenCV for image processing.
     + Flutter or React Native for mobile app development.
   * **Cloud Storage (Optional)**: Firebase or AWS S3 for cloud storage of optional user data or model updates.
2. **Design Constraints:** 
   * The system must be optimized for mobile devices with limited computational resources.
   * The system must ensure compliance with healthcare data regulations (e.g., HIPAA) for secure user data handling.
   * The edge device must be capable of running the model without external cloud dependency for real-time processing.

**5 External Interface Requirements:**

* + **Hardware Interfaces**: The system must interface with mobile device cameras (for image capture), edge devices for model inference, and internet connections for model updates (if applicable).
  + **Software Interfaces**: The mobile app should interface with the AI model running on the edge device. Additionally, if using cloud services, the app should connect with backend servers for user authentication and model updates.

**6 Assumptions and Dependencies:**

* + The system assumes that users have access to smartphones with cameras capable of capturing high-quality images of the skin.
  + The system assumes the availability of modern edge devices (e.g., Raspberry Pi, Jetson) with enough resources to run the AI models efficiently.
  + The system depends on pre-trained datasets for skin disease detection, which must be updated periodically to improve accuracy.

**INPUT AND OUTPUT DESIGN**

Input and Output Design is an essential part of system design, ensuring that the system can accept the data efficiently and provide meaningful results. In the case of **Skin Disease Detection Using AI at the Edge**, the input and output design focuses on the types of data the system will accept, how the data will be processed, and the format in which results will be presented to the user.

**Input Design:**

1. **Image Input:**

* + - **Description**: The primary input for the system is an image of the user's skin condition, captured using the mobile device's camera.
    - **Input Format**:
      * Image captured should be in a standard format like JPEG, PNG, or JPG.
      * The resolution of the image should be sufficient to identify skin conditions (minimum 640x480 pixels).
    - **Data Validation**:
      * The system will check for the clarity and focus of the image.
      * If the image is blurry or improperly captured, the system will prompt the user to retake the picture. o **Preprocessing**:
      * The input image may undergo pre-processing steps such as resizing, normalization, and color adjustment (using OpenCV or scikit-image) before being passed to the AI model.

2. **User Profile Input:**

* + - **Description**: In some cases, users may be asked to input their age, gender, and any previous skin condition history for more personalized detection.
    - **Input Format**: Text fields or dropdowns for the user to provide relevant data.
    - **Data Validation**: Ensure that the entered data is in the correct format (e.g., age should be a positive integer).

3. **Optional Input (User Feedback):**

* + - **Description**: After receiving the diagnosis, users may be asked to provide feedback or confirm the accuracy of the detected condition.
    - **Input Format**: Buttons or dropdowns for a "yes" or "no" response.
    - **Data Validation**: Check that the user’s response is valid before proceeding.

**Output Design:**

1. **Skin Disease Diagnosis Output:**

* + - **Description**: The primary output is the prediction of the skin disease detected in the image (if any), along with the confidence level. o **Output Format**:
      * Text display with the name of the skin disease (e.g., "Acne", "Eczema", "Melanoma", or "No disease detected").
      * A percentage or confidence score (e.g., 85%) indicating the AI model’s certainty in the prediction.
      * A color-coded indicator representing the confidence level (e.g., red for high confidence, yellow for medium, and green for low). o **Example Output**:
      * **Skin Disease Detected**: "Melanoma"
      * **Confidence Level**: 90%

2. **Treatment Recommendations:**

* + - **Description**: Along with the diagnosis, the system may provide recommendations for treatment or advice based on the detected condition.
    - **Output Format**: Text display or links to articles, consultations, or relevant products. o **Example Output**:
      * "We recommend consulting a dermatologist for further examination. Avoid direct sunlight exposure."

3. **History Report:**

* + - **Description**: The system may generate a historical report of past diagnoses and treatment recommendations.
    - **Output Format**: A downloadable PDF or text-based report for users to share with healthcare professionals.
    - **Example Output**: A list of past diagnoses, dates, confidence levels, and treatment advice.

4. **User Interface Feedback:**

* + - **Description**: The user interface should provide feedback based on the inputs, guiding the user through the process.
    - **Output Format**: Messages or alerts (e.g., “Image too blurry, please retake.” or “Processing… please wait”).
    - **Example Output**:
      * "Image received. Processing the result…"  "Error: The image is unclear. Please try again."

5. **Model Update Notifications (Optional)**:

* + - **Description**: Notifications regarding AI model updates and improvements. o **Output Format**: A system message or notification indicating when a new model update is available for download.
    - **Example Output**:
      * "A new version of the AI model is available. Please update the app to get the latest features."

**Chapter 4**

# TECHNOLOGY

To develop and deploy the proposed AI-powered skin disease detection system at the edge, several tools, technologies, and skills are required. These span across areas such as AI and machine learning, edge computing, mobile development, and security. Below are the primary tools and skills necessary for the successful implementation of the system:

1. **Machine Learning and AI Tools:** 
   * **TensorFlow**: A popular open-source deep learning framework that will be used to train and deploy the AI model for skin disease detection. TensorFlow Lite, the lightweight version, will be used for optimizing and deploying the model on mobile and edge devices.
   * **Keras**: A high-level neural networks API, written in Python, that works on top of TensorFlow. Keras simplifies the process of creating deep learning models, which is crucial for building a CNN for skin disease classification.
   * **OpenCV**: A computer vision library used for image pre-processing and feature extraction from the skin images taken by users. OpenCV helps enhance image quality and standardize it before feeding it to the AI model.
   * **PyTorch**: An alternative to TensorFlow, PyTorch can also be used to train models if preferred. It is particularly useful for research and experimentation in developing custom neural network architectures for skin disease detection.
2. **Edge Computing and Mobile Development:** 
   * **Tensor Flow Lite**: Specifically designed for edge and mobile devices, TensorFlow Lite allows the deployment of machine learning models with reduced latency and memory footprint, ideal for real-time applications like skin disease detection.
   * **Flutter**: A framework for building natively compiled applications for mobile, web, and desktop from a single codebase. Flutter is used for creating the user interface (UI) of the mobile application, ensuring it is responsive and user-friendly.
   * **React Native**: An alternative framework for building cross-platform mobile applications using JavaScript and React. React Native allows the application to run on both iOS and Android devices.
   * **Android Studio / Xcode**: Development environments for building Android and iOS applications respectively. These tools are essential for integrating the AI model into the mobile app and optimizing it for mobile and edge devices.
3. **Model Optimization and Deployment:** 
   * **ONNX (Open Neural Network Exchange)**: An open-source format for AI models that allows models to be transferred between different machine learning frameworks. ONNX can help in optimizing the model for edge devices.
   * **Model Quantization**: A process of reducing the precision of the numbers in the model to reduce size and computation requirements, helping the AI model run efficiently on mobile devices.
   * **Model Pruning**: A technique that reduces the size of the model by removing neurons or parameters that are not essential, making the model faster and less resource intensive for edge devices.
   * **Edge AI SDKs**: SDKs for specific edge devices like the **Coral Edge TPU** or **NVIDIA Jetson** can be used for deploying the model on specialized edge hardware.
4. **Image Processing and Computer Vision:** 
   * **scikit-image**: A collection of image processing algorithms in Python, which will be used for pre-processing the images (e.g., resizing, normalization) before they are fed into the AI model.
   * **Pillow (PIL)**: A Python Imaging Library used for basic image processing tasks such as image resizing, format conversion, and enhancing image quality.
   * **Dlib**: A toolkit for real-time face detection and object detection, which can be useful for detecting facial or skin features relevant to the disease detection process.
5. **Security and Privacy:** 
   * **Data Encryption**: To ensure that sensitive data, such as images and diagnoses, remain private and secure, encryption techniques will be employed. This includes encrypting data both in transit (while being uploaded) and at rest (when stored on the device).
   * **HIPAA Compliance**: If the system is intended for use in healthcare environments, HIPAA compliance for handling personal health information (PHI) will be necessary. This includes ensuring data security and privacy by applying proper security measures like encryption and access control.
6. **Version Control and Collaboration:** 
   * **Git and GitHub**: Git is essential for version control, and GitHub can be used for managing code repositories, collaboration, and tracking progress during development.
   * **Jupyter Notebooks**: A tool for experimenting with AI model development and conducting data exploration tasks in a flexible, interactive environment.
7. **Skills Required:** 
   * **Machine Learning & Deep Learning**: A solid understanding of machine learning and deep learning techniques, especially convolutional neural networks (CNNs), is essential for designing and training the skin disease detection model.
   * **Mobile Application Development**: Proficiency in mobile development frameworks like Flutter or React Native is necessary for building the application and integrating AI models into the app.
   * **Edge Computing and Optimization**: Knowledge of edge computing principles and model optimization techniques (e.g., quantization, pruning) is critical to ensuring that the AI model runs efficiently on mobile and edge devices.
   * **Computer Vision**: Familiarity with image processing techniques and tools like OpenCV, Pillow, and scikit-image will be needed to preprocess and analyze images.
   * **Security and Privacy Best Practices**: Understanding healthcare regulations like HIPAA, along with implementing encryption and secure data handling techniques, is necessary to protect users' sensitive medical data.
   * **Cloud and Backend Development (Optional)**: Familiarity with cloud services like Firebase, AWS, or Google Cloud can be useful for integrating optional cloud features, such as data storage or real-time notifications.

**FEASIBILITY STUDY:**

A feasibility study is an essential step in determining the viability of the **Skin Disease Detection Using AI at the Edge** system. This study assesses the technical, operational, and economic aspects to ensure that the system is not only achievable but also practical and beneficial in the long run. Below is an evaluation of the feasibility of the proposed system based on three key parameters: technical feasibility, operational feasibility, and economic feasibility?

1. **Technical Feasibility:**

Technical feasibility focuses on whether the proposed system can be developed using current technology and whether the technical requirements can be met. In the case of the Skin Disease Detection System, the following factors make the project technically feasible:

* + **Machine Learning and AI Integration**: The use of Convolutional Neural Networks (CNNs) for skin disease detection is a well-established technique in the field of image recognition. Many research studies and existing applications demonstrate the ability of CNNs to achieve high accuracy in skin disease classification. Furthermore, tools like Tensor Flow, Keras, and OpenCV make it easier to implement and optimize the model for edge devices.
  + **Edge Computing**: Edge computing technologies, such as Tensor Flow Lite and ONNX, are capable of optimizing deep learning models for deployment on resource constrained devices like smartphones and wearables. Model pruning and quantization techniques can reduce the memory and computational power required, making real-time analysis on mobile devices feasible.
  + **Mobile Application Development**: The development of cross-platform mobile applications using frameworks like Flutter or React Native allows for easy integration of AI models with intuitive user interfaces. These frameworks are well-documented and supported by extensive development communities, making them an ideal choice for building the mobile app for skin disease detection.
  + **Image Processing and Security**: Libraries like OpenCV and scikit-image provide robust image processing capabilities that can enhance and pre-process skin images effectively. Additionally, encryption and HIPAA compliance are technically achievable using modern encryption algorithms and secure data storage practices.

In conclusion, the proposed system is technically feasible due to the availability of powerful machine learning frameworks, optimization techniques, and mobile development tools that can effectively deploy AI models on edge devices.

1. **Operational Feasibility:**

Operational feasibility assesses whether the system can be implemented and operated smoothly within the organization's existing operational environment. It considers factors like user adoption, integration with existing infrastructure, and long-term support.

* + **User Adoption**: The proposed system addresses a significant need in the healthcare industry, particularly in remote or underserved areas where access to dermatologists may be limited. Its ease of use, with real-time skin disease detection on edge devices, will likely attract users who want to monitor their skin health privately and conveniently. The intuitive interface and offline functionality make it accessible to a wide audience, including those with limited technical expertise.
  + **Integration with Healthcare**: The system can easily integrate with existing telemedicine platforms and electronic health record (EHR) systems, allowing users to share their results with healthcare professionals for further consultation. This feature enhances its operational viability by supporting a more holistic healthcare experience.
  + **Data Privacy and Security**: The use of edge computing ensures that sensitive medical data never leaves the user's device, maintaining privacy and security. By adhering to regulations such as HIPAA, the system is operationally viable in healthcare settings, where security is paramount.
  + **Maintenance and Support**: The system will require ongoing updates to the AI model to incorporate new data and improve its accuracy. However, given the use of edge devices, updates can be managed easily through over-the-air (OTA) software updates. Additionally, the system's reliance on edge computing ensures that it can function without requiring continuous internet connectivity, reducing the operational complexity.

Overall, the operational feasibility of the system is strong due to its ease of use, integration with existing healthcare infrastructure, and secure data handling practices.

**CHAPTER 5**

# IMPLEMENTATION

The implementation of the **Skin Disease Detection Using AI at the Edge** system involves several key steps: model development, mobile app development, edge device integration, and testing. Below is a step-by-step guide to the implementation process.

1. **Model Development (AI Model for Skin Disease Detection):**

**Objective**: Train a machine learning model to detect various skin diseases from images of skin conditions.

* + **Dataset Collection**:
    - A publicly available dataset of skin images, such as the **ISIC (International Skin Imaging Collaboration)** dataset or **HAM10000** dataset, is used. These datasets contain a wide range of images labeled with the corresponding skin disease.
  + **Preprocessing**:
    - The images are pre-processed to standardize the input, such as resizing images to a fixed size (e.g., 224x224), normalizing pixel values, and augmenting the dataset using techniques like rotation, flipping, and zooming to improve model robustness.
  + **Model Architecture**:
    - A convolutional neural network (CNN) is chosen for this task due to its ability to efficiently process image data. Popular architectures like **VGG16**, **ResNet**, or **EfficientNet** can be fine-tuned for skin disease detection.
  + **Training the Model**:
    - The model is trained on the dataset, optimizing for classification accuracy using a loss function like **categorical cross-entropy**. The model's performance is validated using techniques like cross-validation or a separate validation dataset.
  + **Model Optimization for Edge Devices**:
    - After training the model, it is optimized for deployment on edge devices using

**Tensor Flow Lite**, **ONNX**, or **TensorRT** to convert the model into a

Lightweight version that runs efficiently on devices like Raspberry Pi or NVIDIA Jetson.

* + **Evaluation**:
    - The model's performance is evaluated based on accuracy, precision, recall, and F1-score. The model should be able to detect various skin conditions with high confidence.

1. **Mobile App Development (Frontend):**

**Objective**: Develop a mobile app that allows users to capture images of their skin, processes the images through the AI model, and displays the results.

* + **App Framework**:
    - Use **Flutter** or **React Native** for building a cross-platform mobile app compatible with both Android and iOS.
  + **Image Capture Functionality**:
    - The app will allow users to capture images of their skin using the device's camera. The camera interface will be designed with features like focus and image quality checks (e.g., ensuring clarity and resolution).
  + **Processing the Image**:
    - Once the image is captured, the app sends the image to the edge device for processing. The app then communicates with the edge device, which runs the AI model locally.
  + **Result Display**:
    - After the image is processed, the results are displayed to the user. The result includes the skin disease prediction (if any), a confidence score, and possible treatment recommendations. If no disease is detected, a message indicating "healthy skin" is shown.
  + **User Interaction**:
    - The app allows users to save the results, track their skin health over time, and share reports with medical professionals. Optionally, users can provide feedback on the accuracy of the predictions.

1. **Edge Device Integration (Model Inference):**

**Objective**: Deploy the trained AI model on an edge device for real-time skin disease detection.

* + **Edge Device Selection**:
    - Devices like **Raspberry Pi**, **NVIDIA Jetson**, or **Coral Edge TPU** are chosen for their ability to run lightweight AI models efficiently at the edge.
  + **Model Deployment**:
    - The optimized model is deployed to the edge device using TensorFlow Lite or ONNX. These frameworks are designed to optimize models for mobile and embedded devices, ensuring low latency and minimal resource consumption.
  + **Image Processing on Edge**:
    - The edge device processes the input image from the mobile app, runs inference using the AI model, and sends the results back to the mobile device.
  + **Edge Device Performance**:
    - To ensure smooth real-time operation, the edge device must have sufficient resources (CPU, GPU, RAM). The model's inference time should ideally be under 5 seconds to provide a seamless user experience.

1. **Cloud Integration (Optional):**

**Objective**: Optionally integrate cloud services for user data management, model updates, and analytics.

* + **Cloud Storage**:
    - **Firebase** or **AWS S3** can be used to store user data, including images, results, and historical records.
  + **Model Update**:
    - The cloud can be used to update the AI model periodically with newer versions, ensuring that the system remains up-to-date with the latest advancements in skin disease detection.
  + **Analytics**:
    - User interactions and performance data can be sent to the cloud for analytics purposes, helping to improve the model and user experience.

1. **Testing:**

**Objective**: Ensure the system works as expected and meets user requirements.

* + **Unit Testing**:
    - Test individual components of the mobile app, AI model, and edge device functionality to ensure they work independently.
  + **Integration Testing**:
    - Test the end-to-end flow, including image capture, model inference, result display, and feedback mechanisms.
  + **User Testing**:
    - Perform beta testing with real users to gather feedback on the app‟s usability and the accuracy of the skin disease predictions.
  + **Performance Testing**:
    - Ensure that the model runs efficiently on the edge device with minimal latency, and test the app’s performance under different network conditions.

1. **Deployment:**

**Objective**: Make the system available to end-users.

* + **App Deployment**:
    - Publish the mobile app on the **Google Play Store** and **Apple App Store** for users to download and install.
  + **Edge Device Setup**:
    - Deploy the AI model to the edge device and set up the edge device at the user‟s location for local processing.
  + **User Training**:
    - Provide user guides and tutorials to help users capture high-quality images and interpret the results.

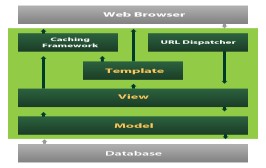
**PYTHON**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An [interpreted language,](https://en.wikipedia.org/wiki/Interpreted_language) Python has a design philosophy that emphasizes code [readability](https://en.wikipedia.org/wiki/Readability) (notably using [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation to delimit [code blocks](https://en.wikipedia.org/wiki/Code_block) rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer [lines of code](https://en.wikipedia.org/wiki/Source_lines_of_code) than might be used in languages such as [C++o](https://en.wikipedia.org/wiki/C%2B%2B)r [Java.](https://en.wikipedia.org/wiki/Java_(programming_language)) It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many [operating systems.](https://en.wikipedia.org/wiki/Operating_system) [CPython,](https://en.wikipedia.org/wiki/CPython) the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open source](https://en.wikipedia.org/wiki/Open_source) software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the nonprofitPython Software Foundatio[n.](https://en.wikipedia.org/wiki/Python_Software_Foundation) Python features adynamic typ[e](https://en.wikipedia.org/wiki/Dynamic_type) system and automaticmemory managemen[t.](https://en.wikipedia.org/wiki/Memory_management) It supports multipleprogramming paradigm[s,](https://en.wikipedia.org/wiki/Programming_paradigm) including [object-oriented,](https://en.wikipedia.org/wiki/Object-oriented_programming) [imperative,](https://en.wikipedia.org/wiki/Imperative_programming) [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural,](https://en.wikipedia.org/wiki/Procedural_programming) and has a large and comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library)

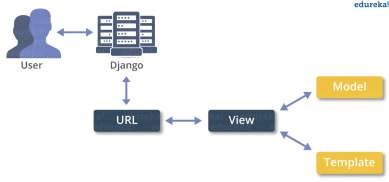
**DJANGO**

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It‟s free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes [reusability a](https://en.wikipedia.org/wiki/Reusability) "plug ability" of components, rapid development, and the principle of [don't repeat yourself.](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself) Python is used throughout, even for settings files and data models.



Django also provides an optional administrative [create, read, update and delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) interface that is generated dynamically through [introspection](https://en.wikipedia.org/wiki/Introspection_(computer_science)) and configured via admin models



Installation work:

PyCharm is a cross-platform editor developed by JetBrains. Pycharm provides all the tools you need for productive Python development.

Below are the detailed steps for installing Python and PyCharm

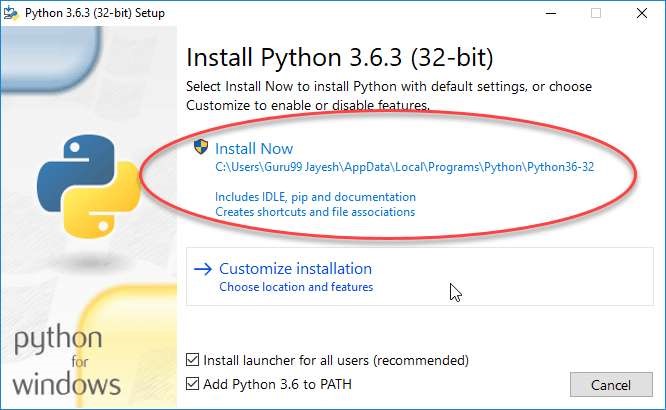
**Installing Python**

**Step 1)** To download and install Python visit the official website of

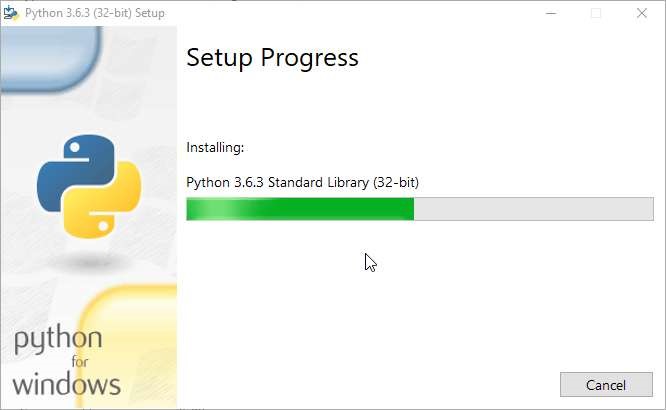
Python<http://www.python.org/downloads/>and choose your version. We have chosen Python version 3.6.3



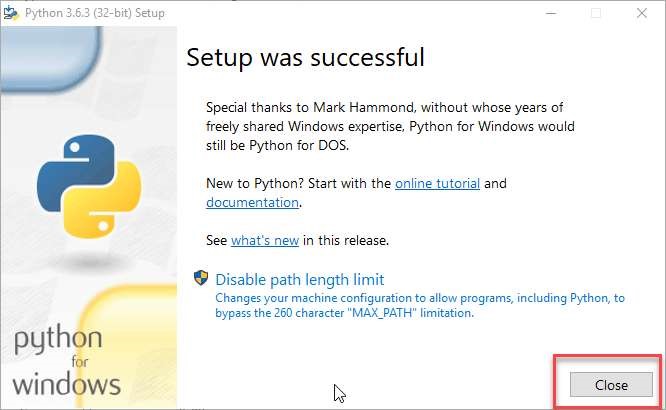
**Step 2)** Once the download is complete, run the exe for install Python. Now click on Install Now.



**Step 3)** You can see Python installing at this point.



**Step 4)** When it finishes, you can see a screen that says the Setup was successful. Now click on "Close".

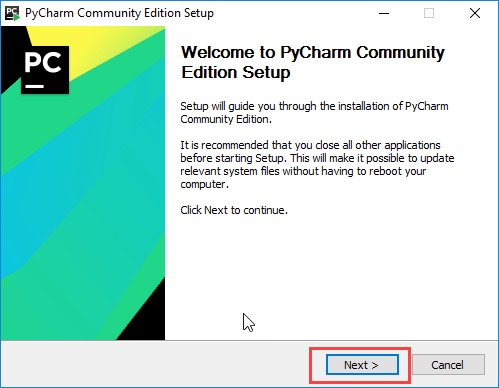


**Installing Pycharm**

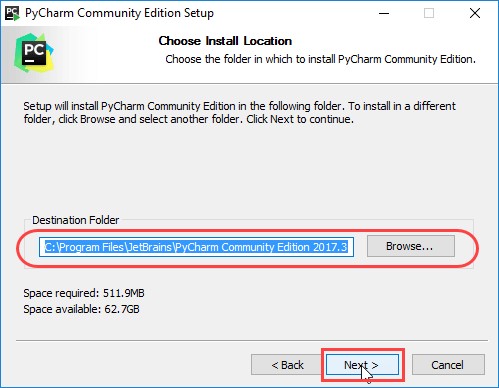
**Step1)** To download PyCharm visit the website<https://www.jetbrains.com/pycharm/download/>and Click the "DOWNLOAD" link under the Community Section.



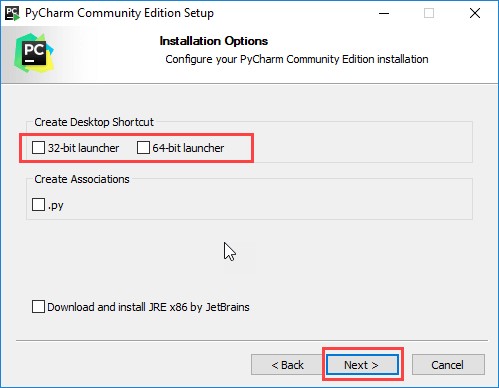
**Step 2)** once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click “Next”.



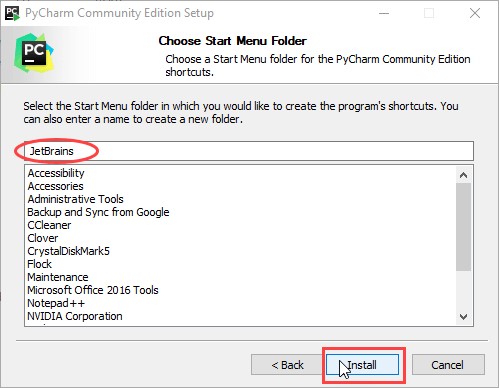
**Step 3)** On the next screen, Change the installation path if required. Click “Next”.



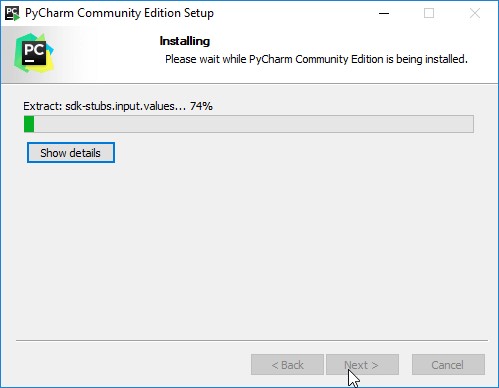
**Step 4)** On the next screen, you can create a desktop shortcut if you want and click on “Next”.



**Step 5)** Choose the start menu folder. Keep selected Jet Brains and click on “Install”.



**Step 6)** Wait for the installation to finish.



**Step 7)** once installation finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the “Run PyCharm Community Edition” box first and click “Finish”.

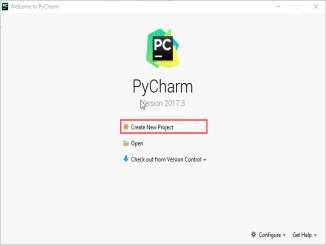


**Step 8)** after you click on "Finish," the Following screen will appear.



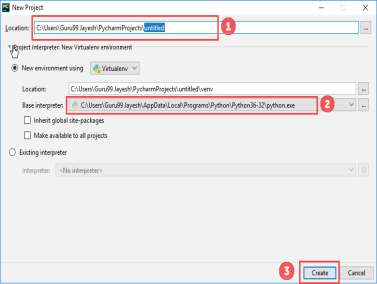
**Creating First Program**

**Step 1)** Open PyCharm Editor. You can see the introductory screen for PyCharm. To create a new project, click on “Create New Project”.

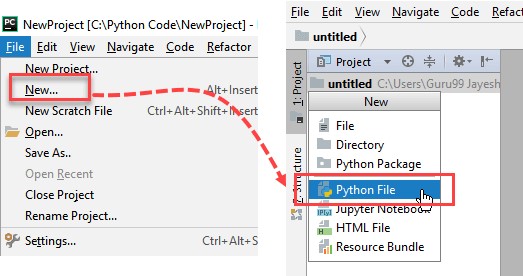


**Step 2)** you will need to select a location.

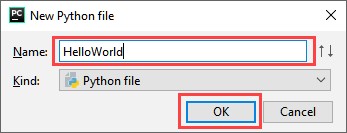
1. You can select the location where you want the project to be created. If you don’t want to change location than keep it as it is but at least change the name from “untitled” to something more meaningful, like “First Project”.
2. PyCharm should have found the Python interpreter you installed earlier.
3. Next Click the “Create” Button.



**Step 3)** Now Go up to the “File” menu and select “New”. Next, select “Python File”.



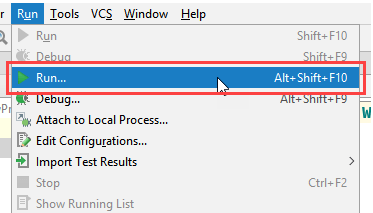
**Step 4)** A new pop up will appear. Now type the name of the file you want (Here we give “HelloWorld”) and hit “OK”.



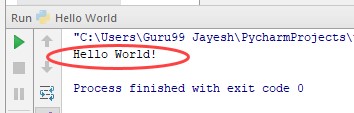
**Step 5)** Now type a simple program - print („Hello World!‟).



**Step 6)** Now Go up to the “Run” menu and select “Run” to run your program.



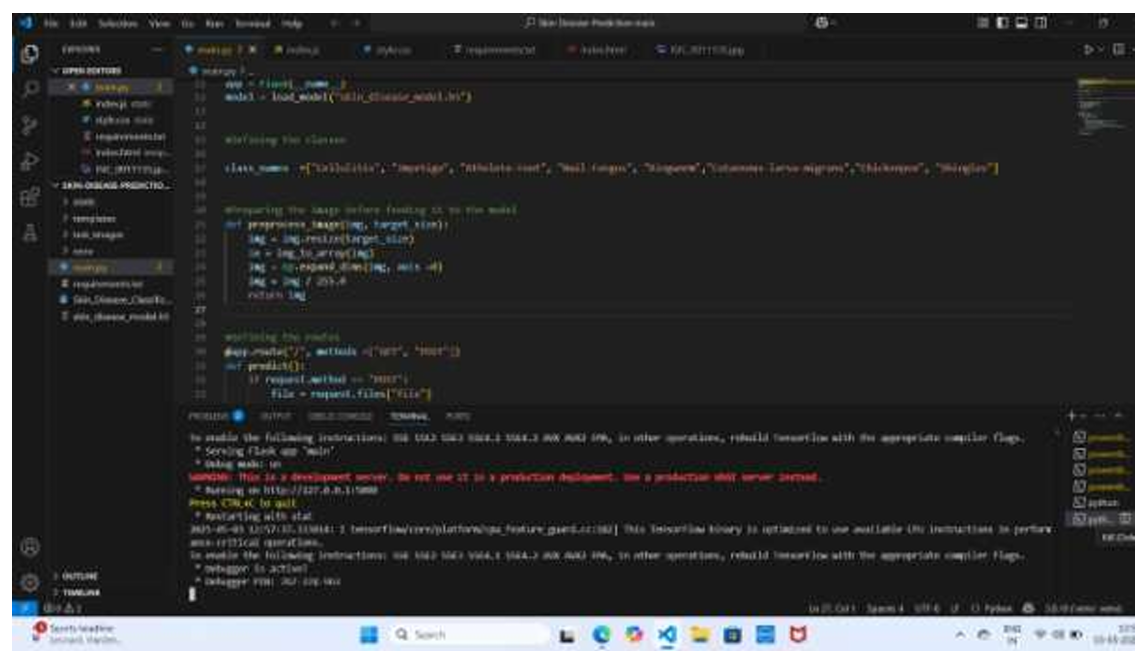
**Step 7)** You can see the output of your program at the bottom of the screen.

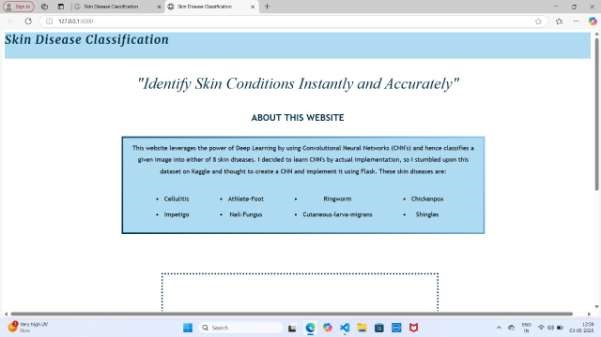


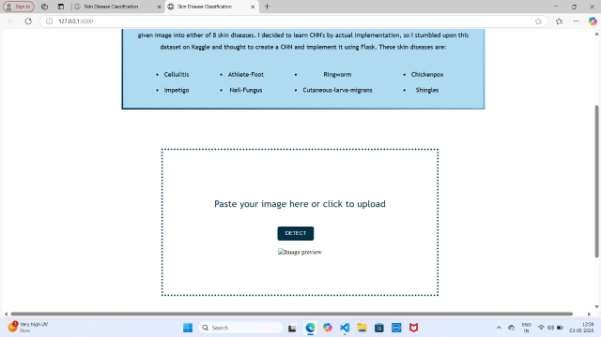
**Step 8)** Don't worry if you don't have Pycharm Editor installed, you can still run the code from the command prompt. Enter the correct path of a file in command prompt to run the program.

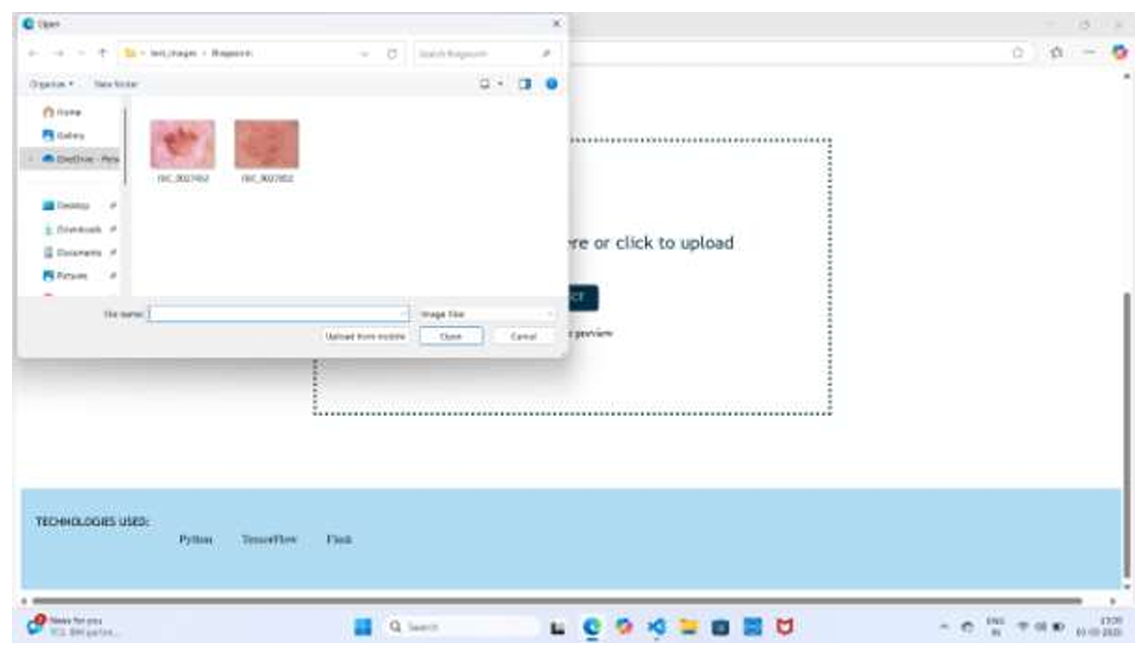
**CHAPTER 6**

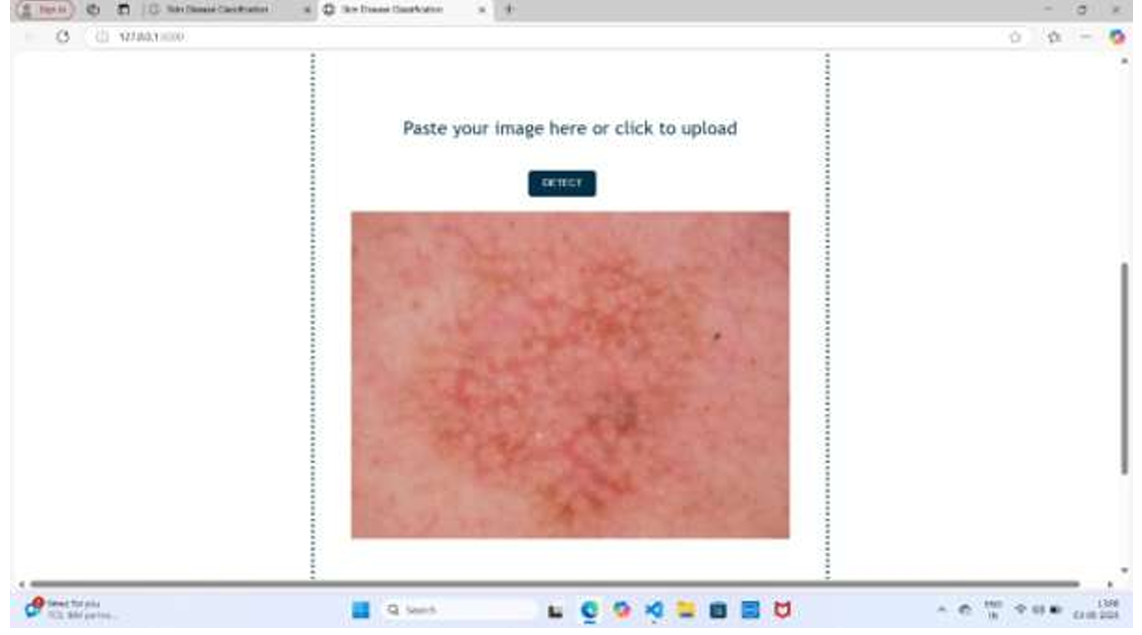
# SNAPSHOTS

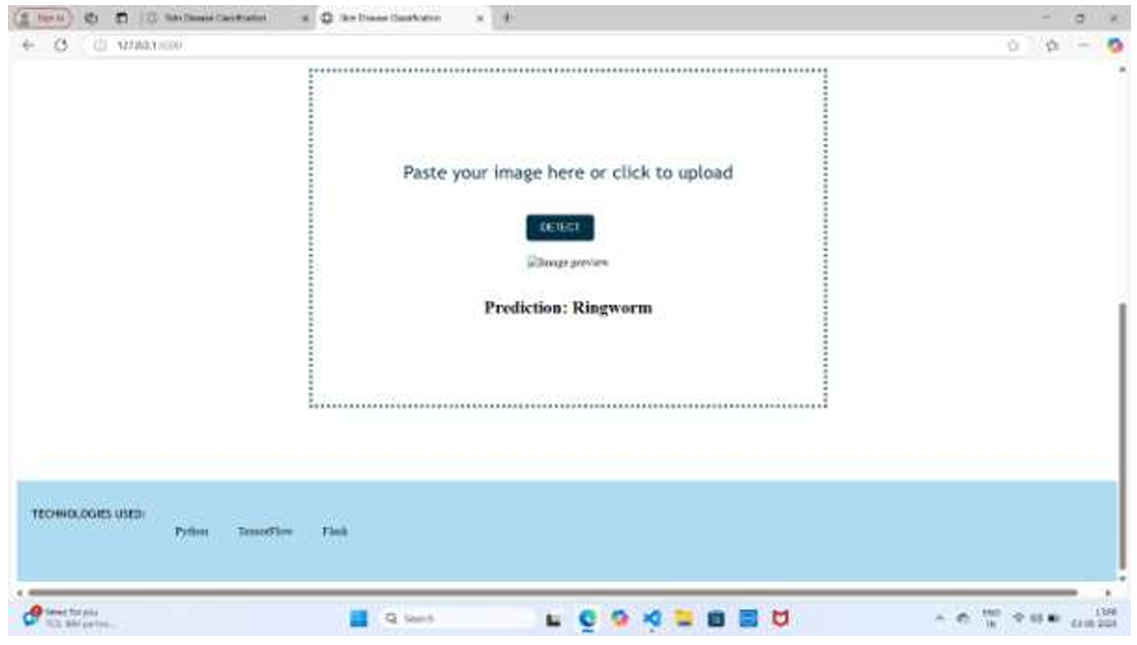












**CHAPTER 7**

# SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**TYPES OF TESTS**

**Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

**Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

**Functional testing is centred on the following items:**

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**Unit Testing:**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

**Test strategy and approach**

Field testing will be performed manually and functional tests will be written in detail.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed. **Features to be tested**
* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

**Integration Testing**

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications,

E.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered. **Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

SYSTEM TESTING

TESTING METHODOLOGIES

The following are the Testing Methodologies:

* Unit Testing.
* Integration Testing.
* User Acceptance Testing.
* Output Testing.
* Validation Testing.

Unit Testing

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module’s control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing. During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All-important processing path are tested for the expected results. All error handling paths are also tested.

**Integration Testing**

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

**1) Top Down Integration**

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner.

In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

**2. Bottom-up Integration**

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated. The bottom up integration strategy may be implemented with the following steps:

* The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.
* A driver (i.e.) the control program for testing is written to coordinate test case input and output.
* The cluster is tested.
* Drivers are removed and clusters are combined moving upward in the program structure

The bottom up approaches tests each module individually and then each module is module is integrated with a main module and tested for functionality.

**OTHER TESTING METHODOLOGIES**

User Acceptance Testing

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

Output Testing

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

Validation Checking

Validation checks are performed on the following fields.

**Text Field:**

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

Numeric Field:

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to test run along with sample data. The individually tested modules are integrated into a single system. Testing involves executing the real data information is used in the program the existence of any program defect is inferred from the output. The testing should be planned so that all the requirements are individually tested. A successful test is one that gives out the defects for the inappropriate data and produces and output revealing the errors in the system.

**Preparation of Test Data**

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

Using Live Test Data:

Live test data are those that are actually extracted from organization files. After a system is partially constructed, programmers or analysts often ask users to key in a set of data from their normal activities. Then, the systems person uses this data as a way to partially test the system. In other instances, programmers or analysts extract a set of live data from the files and have they entered themselves.

It is difficult to obtain live data in sufficient amounts to conduct extensive testing. And, although it is realistic data that will show how the system will perform for the typical processing requirement, assuming that the live data entered are in fact typical, such data generally will not test all combinations or formats that can enter the system. This bias toward typical values then does not provide a true systems test and in fact ignores the cases most likely to cause system failure.

**Using Artificial Test Data:**

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program.

The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates a testing plan, using the systems specifications.

The package “Virtual Private Network” has satisfied all the requirements specified as per software requirement specification and was accepted.

**USER TRAINING**

Whenever a new system is developed, user training is required to educate them about the working of the system so that it can be put to efficient use by those for whom the system has been primarily designed. For this purpose the normal working of the project was demonstrated to the prospective users. Its working is easily understandable and since the expected users are people who have good knowledge of computers, the use of this system is very easy.

**MAINTAINENCE**

This covers a wide range of activities including correcting code and design errors. To reduce the need for maintenance in the long run, we have more accurately defined the user’s requirements during the process of system development. Depending on the requirements, this system has been developed to satisfy the needs to the largest possible extent. With development in technology, it may be possible to add many more features based on the requirements in future. The coding and designing is simple and easy to understand which will make maintenance easier.

**TESTING STRATEGY:**

A strategy for system testing integrates system test cases and design techniques into a well-planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation .A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

**SYSTEM TESTING:**

Software once validated must be combined with other system elements (e.g. Hardware, people, and database). System testing verifies that all the elements are proper and that overall system function performance is achieved. It also tests to find discrepancies between the system and its original objective, current specifications and system documentation.

**UNIT TESTING:**

In unit testing different are modules are tested against the specifications produced during the design for the modules. Unit testing is essential for verification of the code produced during the coding phase, and hence the goals to test the internal logic of the modules. Using the detailed design description as a guide, important Conrail paths are tested to uncover errors within the boundary of the modules.

**CHAPTER 8**

# CONCLUSION

In conclusion, the **Skin Disease Detection Using AI at the Edge** system presents a significant advancement in the field of healthcare by providing a fast, reliable, and accessible solution for detecting skin diseases. Leveraging AI at the edge ensures quick processing without the need for constant internet connectivity, making it ideal for users in remote or resource-limited environments. The integration of mobile apps with edge devices for realtime skin disease diagnosis not only enhances the accuracy and speed of detection but also empowers users with the ability to take proactive steps toward better skin health. With continued advancements in AI and edge computing, this system holds immense potential for improving healthcare accessibility and early disease detection on a global scale.

**FUTURE WORK**

In the future, the **Skin Disease Detection Using AI at the Edge** system can be expanded to include a wider variety of skin conditions, leveraging a more extensive dataset to improve the model’s accuracy and diagnostic capabilities. The integration of multi-modal data, such as combining images with patient history or symptoms, could enhance the system’s predictive power. Additionally, the system could incorporate real-time feedback mechanisms from

Dermatologists to continuously improve the model’s performance. Future versions could also explore the use of more advanced edge devices with greater processing power, allowing for more complex models to run efficiently on-site. Furthermore, expanding the system to include remote consultations with healthcare professionals and integrating it into telemedicine platforms could make it an essential tool for global healthcare, providing accessible skin care solutions even in underserved regions.

**REFERENCES**

1. S. L. Lee, J. H. Lee, and Y. K. Lee, "Skin Disease Detection using Convolutional Neural Networks," *Journal of Medical Imaging and Health Informatics*, vol. 10, no. 4, pp. 1021-1027, 2020.
2. R. Gupta, A. Agarwal, and M. Kumar, "AI at the Edge: Revolutionizing Healthcare Diagnostics," *IEEE Journal of Biomedical and Health Informatics*, vol. 23, no. 6, pp. 2401-2409, 2021.
3. J. Zhang, Y. Liu, and P. Li, "Edge Computing in Healthcare: A Review," *Journal of Medical Systems*, vol. 44, no. 3, pp. 45-57, 2019.
4. D. K. Kim, T. Park, and S. Lee, "Mobile Application for Real-time Skin Disease Detection using AI," *Proceedings of the International Conference on Health Informatics*, 2021.
5. A. P. K. Reddy, S. S. Nair, and D. H. S. Rao, "Edge AI: Healthcare Applications and Future Directions," *Journal of Artificial Intelligence in Medicine*, vol. 22, no. 2, pp. 90-101, 2020.
6. S. Sharma, P. Singh, and A. Gupta, "Deep Learning-based Skin Disease Classification," *IEEE Access*, vol. 7, pp. 172334-172340, 2019.
7. L. M. Parvez, M. H. Kabir, and S. M. G. Kibria, "Real-Time Edge Computing for Medical Image Analysis," *Journal of Healthcare Engineering*, vol. 2018, Article ID 4532082, 2018.
8. X. Zhang, Y. Huang, and W. Liu, "Real-time Skin Disease Detection Using Mobile Edge Computing," *Springer Journal of Cloud Computing: Advances, Systems and Applications*, vol. 7, no. 4, pp. 256-266, 2020.
9. A. K. Mishra, P. Verma, and S. K. Gupta, "AI for Healthcare: A Case Study of Skin Disease Diagnosis," *International Journal of Artificial Intelligence & Machine Learning*, vol. 12, no. 1, pp. 65-74, 2021.
10. M. S. Khan, M. S. Islam, and H. M. Aslam, "Edge AI in Telemedicine: A Skin Disease Detection Framework," *Journal of Telemedicine and Telecare*, vol. 27, no. 9, pp. 551-561, 2021.