EARLY DETECTION AND CLASSIFICATION OF SKIN DISEASES

21CSE315P - Data Orchestration and Management in Cloud Ecosystems
Mini Project Report

Submitted by

M. Krishna Chaitanya [Reg. No: RA2211028010165]

Hari Prasad. T [Reg. No: RA2211028010177]

P. Sai Yaswanth [Reg. No: RA2211028010191]

Under the Guidance of

Dr. S. Prabakeran

Associate Professor, Department of Networking and Communications

In partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING
with a specialization in Cloud Computing

DEPARTMENT OF NETWORKING AND COMMUNICATIONS
SCHOOL OF COMPUTING
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Under Section 3 of UGC Act, 1956)
SRM NAGAR, KATTANKULATHUR – 603 203
CHENGALPATTU DISTRICT

NOVEMBER 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech, Minor project report titled "**EARLY DETECTION AND CLASSIFICATION OF SKIN DISEASES**" is the bonafide work of **M. Krishna Chaitanya [RA2211028010165], Hari Prasad. T [RA2211028010177] and P. Sai Yaswanth [RA2211028010191]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation based on which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. S. Prabakeran

SUPERVISOR
Associate Professor
Department of Networking and Communications,
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Dept. of Networking and Communications
Kattankulathur 603 203
SRM IST

Dr. M.Lakshmi

PROFESSOR AND HEAD
Department of Networking and Communications,
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

SRM

Department of Networking and Communications

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/ Course : B.Tech

Student Name : **M.Krishna Chaitanya , Hari Prasad.T, P. Sai Yaswanth**

Registration Number : RA221108010165,RA2211028010177,RA2211028010191

Title of Work : **EARLY DETECTION AND CLASSIFICATION OF SKIN DISEASES**

I hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I confirm that all the work contained in this assessment is my own except where indicated, and that I have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g.fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook /University website

I understand that any false claim for this work will be penalized in accordance with theUniversity policies and regulations.

DECLARATION:

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my own work, except where indicated by referring, and that I have followed the good academic practices noted above.

M.Krishna Chaitanya (RA2211028010165) M.Krishna Chaitanya

Hari Prasad.T (RA2211028010177)

P. Sai Yaswanth (RA2211028010191) P. S. Yaswanth.

If you are working in a group, please write your registration numbers and sign with the date forevery student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan,** Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr.T.V. Gopal,** for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman, Professor & Chairperson**, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M . Lakshmi** , Professor and Head, Department of Networking and Communications, School of Computing, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

Our inexpressible respect and thanks to our guide, **Dr. S. Prabakeran**, Associate Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Networking and Communications, Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

M. Krishna Chaitanya [RA2211028010165]

Hari Prasad. T [RA2211028010177]

P. Sai Yaswanth [RA221102801091]

ABSTRACT

The rise of dermatological conditions, combined with the increasing demand for efficient and scalable diagnostic solution. The proposed project introduces a web-based application for the early detection and classification of skin diseases, leveraging a Convolutional Neural Network (CNN) model. For this application, users shall upload images of their specific skin conditions, which, in turn, are going to be processed by Flask server communicating with a pre-trained CNN model fine-tuned to classify six skin diseases. Utilizing Kafka to stream the data enables responsive real-time analysis as data streams and is quickly transmitted with instantaneous feedback. Then, HBase is used to store big data quite efficiently to allow for updating of models as well as more in-depth analyses of the data. This system seeks to give users knowledge instantly concerning possible conditions influencing their skin, enabling early diagnosis and management of diseases associated with the skin to enhance health care outcomes in general.

Keywords: Real-time Data Streaming, Image Classification, Disease Prediction,CNN, Support Vector Machine (SVM), Kafka, HBase, Data Analytics

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER-1

INTRODUCTION TO EARLY DETECTION AND CLASSIFICATION OF SKIN DISEASES

The rapid advancement of technology in healthcare has created new avenues for accessible and proactive health management. With skin diseases being among the most common health concerns worldwide, early detection and classification are crucial for effective treatment. However, traditional diagnostic methods often require in-person consultations, which may not be accessible to everyone due to location, cost, or resource limitations. This project addresses these challenges by developing an AI-driven system for the early detection and classification of skin diseases through an intuitive and accessible web application. This solution aims to bridge the gap between individuals and healthcare specialists by providing a reliable tool for preliminary, real-time diagnostic support. Leveraging a Convolutional Neural Network (CNN), the system is designed to accurately classify various skin conditions based on a large, curated dataset of annotated images, enabling users to receive immediate feedback on their skin health.

To achieve a seamless and efficient pipeline, the project integrates several key technologies. A Python Flask server handles backend processing, ensuring robust communication between the user interface and the image classification model. Apache Kafka is employed for real-time data streaming, facilitating efficient data flow between application layers, while HBase provides scalable, distributed storage that allows the system to manage large datasets effectively, even as data volumes grow. This architecture supports real-time capabilities, making the platform responsive and capable of delivering fast and accurate analysis for users.The web application is designed with a strong emphasis on user accessibility and simplicity. The interface allows users to upload images of their skin conditions quickly, without requiring any technical expertise. Once an image is submitted, the system processes it and delivers a real-time prediction of the potential skin condition, empowering users to make informed health decisions and promoting a proactive approach to skin health management. Beyond basic classification, the project aims to provide valuable insights that can support users in taking timely steps toward medical consultation, ultimately enhancing healthcare accessibility. By transforming skin health analysis into a user-

friendly, AI-assisted experience, this project seeks to make early detection a practical reality, fostering better health outcomes and improved quality of life.

1.1 Motivation:

Skin diseases are prevalent worldwide, yet they are often overlooked due to limited access to specialist healthcare, particularly in low-income and remote areas. Dermatological conditions, if not diagnosed and treated in time, can lead to severe consequences, including persistent discomfort, the progression of diseases, increased healthcare costs, and more complex treatment regimens. In the absence of professional medical advice, many individuals resort to self-medication, which not only risks incorrect treatment but can also exacerbate the condition, leading to potentially life-threatening complications.This project aims to bridge the accessibility gap by providing a reliable, AI-driven solution for real-time, preliminary skin disease classification. By utilizing Convolutional Neural Networks (CNNs) and leveraging a curated image dataset, this tool aspires to offer users a quick assessment of their skin condition. Such a system empowers users to take the first step towards seeking medical intervention, thus preventing conditions from worsening due to delays in diagnosis. The goal is not to replace medical professionals but to function as a supplementary aid that encourages users to pursue appropriate medical care at the earliest.In essence, this project aspires to revolutionize the accessibility of dermatological care, offering a timely and cost-efficient diagnostic alternative that can significantly impact public health outcomes. By encouraging early detection, fostering health awareness, and empowering individuals with accessible technology, this initiative holds the potential to improve the quality of life for millions, especially those residing in areas with limited access to specialist care.

1.2 Problem Statement

` Millions of people worldwide are affected by skin diseases, which often require early diagnosis to ensure effective treatment and prevent further health complications. However, access to dermatologists remains limited, particularly in remote or resource-poor settings, leading to delayed diagnoses and potentially worsened conditions. The lack of accessible, timely skin health assessment tools represents a critical barrier to proactive health management for many individuals. Traditional diagnostic pathways rely heavily on in-person consultations, which are not feasible for everyone due to geographical and financial constraints.

This project addresses these challenges by developing an automated, AI-powered solution for the early detection and classification of common skin diseases. By providing a free, user-friendly tool that leverages advanced machine learning techniques, this solution aims to bridge the accessibility gap and empower users to obtain preliminary assessments of their skin health from the convenience of their own devices. Designed to handle images of various skin conditions, this tool offers fast and reliable feedback, allowing users to gain insights that can encourage timely medical follow-up when needed. With the integration of real-time image processing and classification capabilities, this project seeks to create a scalable and effective resource for proactive skin health management, ultimately contributing to improved health outcomes and increased accessibility to early diagnostic support.

1.3 Objective of the Project:

The primary objective of this project is to develop an accessible and intuitive web application that empowers users to upload images of their skin conditions and receive real-time, preliminary diagnostic results. The solution aims to bridge the gap between individuals and healthcare specialists by offering a reliable, AI-driven tool that provides instant assessments. By harnessing the power of a Convolutional Neural Network (CNN), the application can effectively classify skin conditions with a high degree of accuracy based on a curated dataset.To achieve this, the project integrates several key technologies to build a seamless and efficient pipeline. A Python Flask server is utilized for backend processing, enabling robust communication between the user interface and the image classification model. Kafka is employed for efficient and real-time data streaming, facilitating smooth data flow between the application layers. Meanwhile, HBase offers scalable and distributed storage, allowing the system to manage large datasets effectively while maintaining quick response times.The implementation of this project is focused on ensuring that users receive fast and accurate analysis of their clinical conditions. The web-based interface is designed to be user-friendly, enabling individuals with minimal technical knowledge to navigate and obtain results effortlessly. By providing a simple upload mechanism, users can easily submit images for analysis and receive real-time feedback on potential skin conditions. This approach empowers users to make informed health decisions and fosters a proactive mindset towards skin health management.

1.4 Scope:

This project is dedicated to creating a comprehensive end-to-end system for the real-time classification of common skin diseases through the power of deep learning. The approach begins with data acquisition and preparation, where an extensive dataset containing annotated images for six frequently observed skin diseases is collected and systematically preprocessed. This step is critical, as it includes tasks like image cleaning, augmentation, and precise labeling, which enhance data quality and diversity to optimize model training. A carefully curated dataset allows the model to generalize better, improving its ability to accurately classify various skin conditions. The core of the system lies in model development, involving the design, training, and fine-tuning of a Convolutional Neural Network (CNN). This CNN architecture is tailored to handle intricate patterns in skin lesion images, ensuring high accuracy and reliability in disease classification.A key emphasis is placed on system integration, bringing together front-end and back-end components to create a smooth, user-centered experience. The user interface is designed for simplicity and accessibility, allowing users to upload images of skin lesions with ease. These images are then forwarded to a Python Flask server that processes classification requests, leveraging the trained CNN model to analyze and return results in real time. This real-time capability is essential, offering users instantaneous feedback on potential skin conditions, thus making the experience interactive and responsive. Furthermore, robust error handling mechanisms ensure that users are guided with clear messages if any issues arise during upload or prediction, thereby fostering a user-friendly experience.The system is engineered with scalability and efficiency in mind. It can adapt to increasing data volumes and a growing number of users without compromising performance, making it suitable for long-term deployment in healthcare settings or public health applications. This approach prioritizes not only technical precision but also practical application, bridging healthcare accessibility gaps by enabling early, AI-driven screening. This screening tool can serve as a valuable resource for individuals and healthcare providers alike, promoting proactive health management and potentially leading to earlier detection of skin diseases. By making advanced diagnostics more accessible, this project aims to contribute significantly to public health, encouraging a shift towards preventive care and enhancing overall healthcare outcomes.

1.5 Relevance:

Early detection and accurate classification of skin diseases are essential for effective treatment and prevention. This project addresses a significant healthcare need by providing a user-friendly and accessible tool that can aid in early diagnosis, potentially reducing the burden on healthcare systems and improving patient outcomes.

The established a comprehensive foundation for understanding the critical importance of early detection and classification of skin diseases, particularly in settings where access to healthcare is limited. It has emphasized the high prevalence of skin conditions worldwide and the pressing need for accessible, scalable diagnostic tools that can reach underserved populations. Traditional diagnostic methods, which rely on in-person consultations, present significant barriers in remote or resource-poor areas, often leading to delayed diagnoses and a subsequent decline in patient outcomes. The chapter has underscored how early intervention is essential for preventing the progression of many skin conditions.By outlining the project's objectives, Chapter 1 has introduced the development of an AI-driven web application aimed at democratizing access to skin health assessments. This application leverages advanced image classification techniques using Convolutional Neural Networks (CNNs) to offer users real-time, preliminary diagnoses of skin conditions. The chapter has also detailed the technical approach, including the integration of real-time processing and data storage solutions to handle large volumes of user data efficiently, ensuring the application remains responsive and scalable as demand grows.

Overall, Chapter 1 has built a strong case for the potential impact of this project on healthcare accessibility. With a focus on empowering users to actively manage their skin health through an intuitive, user-friendly platform, this solution aims to bridge the gap between individuals and specialized dermatological care. Ultimately, this project aspires to make early detection more accessible for diverse populations, promoting proactive health management and helping to address the broader challenge of healthcare inequality on a global scale.

CHAPTER-2

LITERATURE SURVEY FOR SKIN DISEASE DETECTION AND CLASSIFICATION USING DEEP LEARNING

In recent years, deep learning has become a transformative force in medical image analysis, offering promising advancements for skin disease classification. This field holds particular relevance, as skin conditions impact not only physical health but also mental well-being, especially for visible areas like the face. By harnessing powerful architectures like Convolutional Neural Networks (CNNs), researchers are achieving diagnostic accuracy levels that can match or even exceed those of skilled dermatologists. With increasing access to clinical images via smartphones and other devices, the integration of these deep learning models into healthcare applications is becoming more feasible and widespread, enabling early and accurate diagnosis directly through digital platforms.

This section reviews critical research contributions in applying CNN and other advanced deep learning architectures, including ResNET and VGG16, to skin disease classification. Additionally, hybrid methods, such as combining CNNs with Local Binary Pattern (LBP) techniques, showcase how both traditional and deep learning approaches can be leveraged to enhance classification accuracy, especially on texture-rich dermatological images. Alongside these advancements, the studies also address ongoing challenges, such as limited datasets, slow processing speeds, and the need for distributed architectures to enable scalability and real-time application. Each research approach provides insights that lay the groundwork for developing a more efficient, scalable skin disease classification model, aiming for real-time data streaming, secure storage, and effective processing solutions

Related Work:

[1] "Studies on Different CNN Algorithms for Face Skin Disease Classification Based on Clinical Images | IEEE Journals & Magazine | IEEE Xplore," *ieeexplore.ieee.org*. https://ieeexplore.ieee.org/document/8720210

Skin problems not only injure physical health but also induce psychological problems, especially for patients whose faces have been damaged or even disfigured. Using smart devices,

most of the people are able to obtain convenient clinical images of their face skin condition. On the other hand, the convolutional neural networks (CNNs) have achieved near or even better performance than human beings in the imaging field. Therefore, this paper studied different CNN algorithms for face skin disease classification based on the clinical images. First, from Xiangya–Derm, which is, to the best of our knowledge, China's largest clinical image dataset of skin diseases, we established a dataset that contains 2656 face images belonging to six common skin diseases [seborrheic keratosis (SK), actinic keratosis (AK), rosacea (ROS), lupus erythematosus (LE), basal cell carcinoma (BCC), and squamous cell carcinoma (SCC)]. We performed studies using five mainstream network algorithms to classify these diseases in the dataset and compared the results. Then, we performed studies using an independent dataset of the same disease types, but from other body parts, to perform transfer learning on our models. Comparing the performances, the models that used transfer learning achieved a higher average precision and recall for almost all structures. In the test dataset, which included 388 facial images, the best model achieved 92.9%, 89.2%, and 84.3% recalls for the LE, BCC, and SK, respectively, and the mean recall and precision reached 77.0% and 70.8%.

Challenges: Very small dataset and file processing will be slow comparing to the file processing in distributed file system.

[2] E. Goceri, "Analysis of Deep Networks with Residual Blocks and Different Activation Functions: Classification of Skin Diseases," ***2019 Ninth International Conference on Image Processing Theory, Tools and Applications (IPTA),*** **Nov. 2019, doi: https://doi.org/10.1109/ipta.2019.8936083.**

Deep convolutional neural networks have been implemented for image classification tasks and achieved promising results in recent years. Particularly, ResNETs have been used since they can eliminate vanishing gradient problem in very deep networks. However, ResNET architectures with different activation functions, batch sizes, number of images in the testing and training stages can cause different results. Therefore, the effect of residual connections and activation functions image classification is still unclear. Also, in the literature, ResNET based models have been trained and tested with data sets having different characteristics. However, to make meaningful evaluations of the results obtained from different ResNET models, the same data sets should be

used. Therefore, in this work, four network models have been implemented to analyze the effect of two activation functions (ReLU and SELU) and residual learning for image classification using the same data sets. To evaluate performances of these models, a real world issue, which is automated skin disease classification from colored digital images, has been handled. Experimental results and comparative analyses indicated that the ResNET with SELU and without residual block yields in the highest validation accuracy (97.01%) for image classification.

Challenges: File processing will be slow comparing to the file processing in distributed file system. Real-time processing of data is not possible

[3] G. Singh, Kalpna Guleria, and S. Sharma, "A Transfer Learning-based Pre-trained VGG16 Model for Skin Disease Classification," pp. 1–6, Dec. 2023, doi: https://doi.org/10.1109/mysurucon59703.2023.10396942.

Skin disorders pose a significant global health risk, impacting millions of individuals and placing a substantial burden on healthcare systems. The accuracy and speed of diagnosis are crucial for effectively managing various conditions. Deep learning models have demonstrated exceptional performance in diverse medical imaging applications, including the categorization of skin diseases. In recent years, the VGG16 deep learning architecture has gained prominence for its ability to extract meaningful features from images. In this study, a VGG16 model has been leveraged to early diagnose skin diseases. This approach involves collecting an extensive dataset comprising images of different skin disorders sourced from an open-source repository "Kaggle". Further, the VGG16 model is then fine-tuned on this collected dataset to learn the distinguishing patterns and characteristics associated with different skin conditions. The evaluation of the model's effectiveness has been done using standard metrics such as precision, recall, F1-score, and accuracy. These metrics assess the model's analytical capabilities in distinguishing between various skin disorders. The proposed deep learning model achieves remarkable accuracy of 90.1%, proving its proficiency in diagnosing a wide range of skin diseases, including those that appear similar. Furthermore, precision, recall, and F1-score have been identified as 0.867, 0.942, and 0.891, respectively. This research contributes to the evolution of computer-aided disease detection, potentially leading to enhanced healthcare outcomes by facilitating early detection and treatment of skin disorders. Nonetheless, continuous refinements and validation on larger, more diverse

datasets are imperative to further enhance the model's accuracy and ability to generalize across various conditions.

[4] G. Cavallaro, M. Riedel, M. Richerzhagen, J. A. Benediktsson, and A. Plaza, "On Understanding Big Data Impacts in Remotely Sensed Image Classification Using Support Vector Machine Methods," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 10, pp. 4634–4646, Oct. 2015, doi: https://doi.org/10.1109/jstars.2015.2458855.

Owing to the recent development of sensor resolutions onboard different Earth observation platforms, remote sensing is an important source of information for mapping and monitoring natural and man-made land covers. Of particular importance is the increasing amounts of available hyperspectral data originating from airborne and satellite sensors such as AVIRIS, HyMap, and Hyperion with very high spectral resolution (i.e., high number of spectral channels) containing rich information for a wide range of applications. A relevant example is the separation of different types of land-cover classes using the data in order to understand, e.g., impacts of natural disasters or changing of city buildings over time. More recently, such increases in the data volume, velocity, and variety of data contributed to the term big data that stand for challenges shared with many other scientific disciplines. On one hand, the amount of available data is increasing in a way that raises the demand for automatic data analysis elements since many of the available data collections are massively underutilized lacking experts for manual investigation. On the other hand, proven statistical methods (e.g., dimensionality reduction) driven by manual approaches have a significant impact in reducing the amount of big data toward smaller smart data contributing to the more recently used terms data value and veracity (i.e., less noise, lower dimensions that capture the most important information). This paper aims to take stock of which proven statistical data mining methods in remote sensing are used to contribute to smart data analysis processes in the light of possible automation as well as scalable and parallel processing techniques. We focus on parallel support vector machines (SVMs) as one of the best out-of-the-box classification methods.

Challenges: SVM is not suitable for large datasets with many features

[5] N. Akmalia, P. Sihombing, and Suherman, "Skin Diseases Classification Using Local Binary Pattern and Convolutional Neural Network," *IEEE Xplore*, 2019. https://ieeexplore.ieee.org/document/8943892

The paper "Skin Diseases Classification Using Local Binary Pattern and Convolutional Neural Network" explores a hybrid approach that combines Local Binary Pattern (LBP) with Convolutional Neural Networks (CNN) to classify various skin diseases. LBP is employed to capture local texture features, which are then fed into a CNN model for further learning and classification. The LBP helps to identify significant patterns within the image, making it easier for the CNN layers to recognize specific textures associated with different skin conditions. This method aims to enhance the model's ability to differentiate between subtle textures present in skin lesions, ultimately improving classification accuracy. By integrating LBP with CNN, the approach leverages both handcrafted and deep learning features, which can be advantageous for medical images with limited datasets. CNN layers process the LBP features, learning higher-level abstractions to categorize skin diseases such as Melanoma, Basal Cell Carcinoma, and more. The paper demonstrates how this dual-method approach can improve diagnostic accuracy

Challenges: The data cleansing process often removes numerous unzoomed images, reducing the dataset size and potentially impacting model performance.

The literature on skin disease classification reflects a strong focus on deep learning architectures, particularly CNNs, for achieving high classification accuracy across various skin conditions. Models such as ResNET and VGG16 have demonstrated the capability to analyze complex patterns and textures within medical images. Additionally, hybrid methods that combine traditional image processing techniques like LBP with CNNs provide promising results, especially for datasets with challenging image textures. limitations such as dataset size, processing speed, and lack of real-time capability remain significant. These challenges underscore the importance of adopting distributed systems, such as Apache Kafka for data streaming and HBase for efficient storage, to build scalable and responsive models. diagnosis, ultimately leading to better patient outcomes and improved accessibility to healthcare solutions.

CHAPTER-3

PROPOSED METHODOLOGY OF REAL-TIME SKIN DISEASE IDENTIFICATION AND CLASSIFICATION USING CNNS

In the digital healthcare landscape, early and accurate detection of skin diseases plays a crucial role in improving patient outcomes and facilitating timely medical interventions. This project introduces an advanced, real-time image classification system for diagnosing skin diseases, utilizing a Convolutional Neural Network (CNN) model integrated with a Python Flask server. The solution is designed to enable users to upload skin-related images for immediate analysis, yielding accurate diagnostic insights powered by a highly optimized CNN model trained on a large, curated dataset of dermatological images.

To enhance responsiveness and scalability, the project leverages Apache Kafka for real-time data streaming, ensuring swift data ingestion and processing. HBase is employed as a robust storage solution, securely handling the vast dataset of images and classification results, thereby supporting data-intensive applications and facilitating continuous improvements to the model over time. This systematic architecture not only delivers real-time, high-accuracy skin disease classifications but also ensures data security and scalability, making it an invaluable tool in modern digital healthcare.

The following sections provide an in-depth exploration of each system component, highlighting how the integration of these technologies results in an efficient, user-centered approach to medical image classification. This innovative pipeline empowers healthcare providers and users with actionable insights, enabling quicker responses and informed decisions that could significantly impact patient health and satisfaction.

3.1 Approach

The project's approach combines a CNN model with a Python Flask server for realizing real-time classification of the uploaded images related to skin diseases by the users. The proposed model will be trained extensively in a highly large dataset containing curated skin disease images

to achieve and optimize high-accuracy classifications. It incorporates Kafka for real-time streaming and uses HBase as a repository for storage and retrieval.

3.2 Advantages

1. High accuracy: This project uses a CNN model fine-tuned to deliver the best possible accuracy in skin disease classification. This network is capable of dealing with complex features that would not be possible for the simplest models to handle, particularly for medical images.
2. Real-Time Performance: Kafka, with the ability to stream data, does provide fast data transfer; users can expect near-instantaneous results of classification. This kind of real-time feedback is important to users who may need to intervene early to prevent a problem from developing.
3. Scalable and Secure Storage: HBase is known to provide a scalable and secured storage solution for the mass amount of dermatological images, classification results, as well as their metadata so that data safety can be ensured and analysis as well as model update may occur over time.
4. User-friendly interface: Web-based interface is intuitive, allowing nontechnical users to upload images easily and receive classification results. This system is cost-effective and accessible, as it can be deployed on local servers or within healthcare facilities, whereas cloud-based AI services face the limitation of limited internet connectivity.
5. Cost effective and accessible: This system is cost-effective and accessible, as it can be deployed on local servers or within healthcare facilities, whereas cloud-based AI services

3.3 Existing System

Traditional Machine Learning Models: Traditional models such as Support Vector Machines (SVM) and Random Forests have been widely applied in skin disease classification. However, the requirement for feature extraction makes it unable to handle complicated image data like it does with CNN.

Mobile Applications for Skin Disease Detection: There are a lot of mobile applications that detect skin diseases using very simple image recognition or heuristic-based algorithms. Indeed,

they make it accessible to the masses, but most of them lack accuracy, and they are not properly integrated with high-performance backend models.

Cloud-Based AI Services: Some cloud-based platforms offer skin disease classification as part of more general AI services. These solutions take advantage of high computing power but are expensive, depend on stable internet connectivity, and have no real-time feedback mechanism.

3.4 Disadvantages

1. Limited scalability, as these models struggle with large image datasets.
2. Manual feature extraction can lead to suboptimal results and reduce accuracy.
3. High costs for users, making them inaccessible in low-resource settings.
4. Lack of real-time capability, which hinders quick diagnosis.

3.5 Data and Resources:

A publicly available dataset of skin disease images from KAGGLE is used to train and evaluate the model.Additional data is collected through user-uploaded images.

3.6 System Architecture

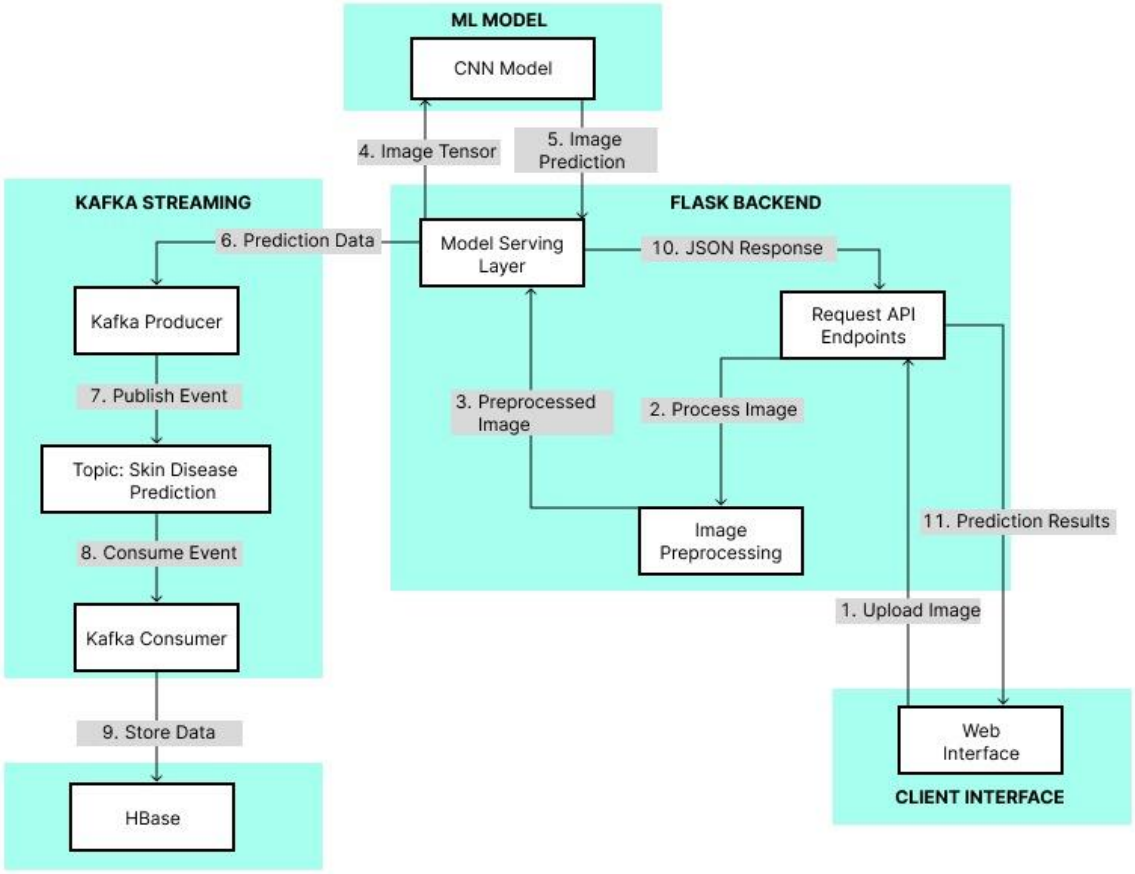


Fig: 3.1 Architecture Diagram

The components of the Fig: 3.1 Architecture Diagram has been listed below

The client interface for this system provides a seamless, user-friendly experience through a web platform designed with HTML, CSS, and JavaScript. Users can upload images of skin lesions, which are sent to the backend via HTTP POST requests. Upon receiving an image, the system initiates an image preprocessing pipeline, essential for preparing the image for analysis by a machine learning model. Preprocessing typically involves resizing, normalization, and feature extraction to ensure that the image conforms to the model's input requirements and maximizes prediction accuracy.

Once preprocessed, the image is passed into a machine learning model, likely a Convolutional Neural Network (CNN) trained on a comprehensive dataset of skin lesion images with known diagnoses. The model then generates a prediction, classifying the lesion and providing insights into the probable skin condition. For asynchronous communication and efficient data management, Kafka streaming is integrated into the system. The image tensor, along with model predictions, is published as events to a Kafka topic titled "skin_disease_predictions," allowing the system to handle data streams efficiently and support real-time processing.

A Flask backend operates as the API layer, exposing RESTful endpoints for handling user requests, including image uploads via HTTP POST and retrieval of prediction results through HTTP GET. Processed images and their corresponding predictions are stored in a database, enabling data persistence for further analysis or future reference. This structured approach ensures a cohesive and scalable solution, facilitating efficient data flow and storage, while delivering an accessible tool for skin disease prediction.

3.7 Techniques and Tools

The project utilizes various technologies, including TensorFlow/Keras for deep learning, OpenCV for image processing, Python Flask for backend development, Kafka for data streaming, HBase for data storage, and CSS for frontend development.

3.8 UML Diagrams

Use case diagram:

In the Unified Modeling Language (UML), a Use-case diagram is a behavioural diagram defined and built out from a Use-case analysis. It is useful to represent graphically what functionalities the system offers with the use of actors and their aims, represented by means of use cases as well as any dependencies amongst the said use cases. The main objective of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. **Fig 3.2** is the use case diagram which illustrates the interactions between two types of users—User and Admin—and the system. Both users can interact with various system functionalities. The User can upload images, which undergo image processing, followed by disease

classification, and the results are stored in the system. The User can also view these results. The Admin has additional responsibilities, including configuring the system, managing databases, monitoring analysis, and exporting reports. The system processes uploaded images and classifies diseases, storing the processed data for further use. This diagram provides a high-level overview of the system's functionality and the roles of different user types.

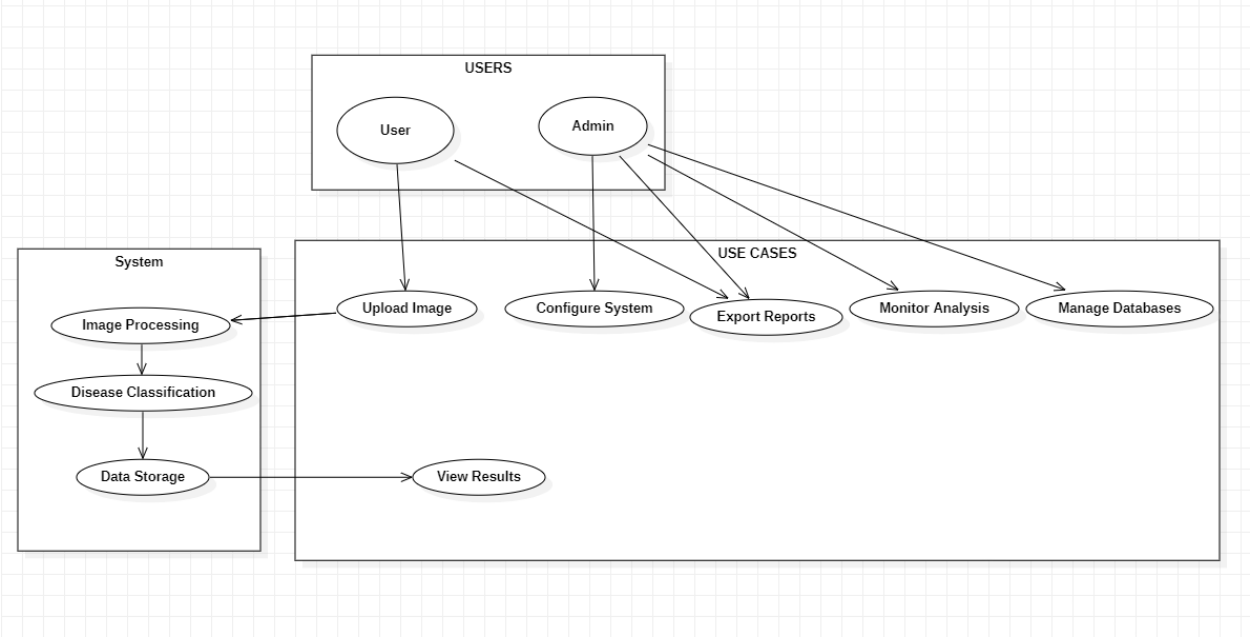


Fig: 3.2 Use Case Diagram

Class Diagram

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that illustrates the architecture of a system by depicting the system's classes, their properties, methods, and the relationships between them. **Fig 3.3** provides the class diagram for the disease classification system, highlighting each component and its interactions within the architecture.

The Web UI class is the primary interface for users, enabling them to interact with the application by uploading images and viewing diagnostic results. It includes essential methods such

as uploadImage() and displayResults(), which facilitate image submission and display classification results in a user-friendly format. The Web UI connects to the Flask layer, which functions as the system's backend, managing incoming image processing requests and retrieving prediction results. This backend component includes methods like processImage() to prepare images for analysis and getResults() to fetch processed outcomes, ensuring smooth communication with the other subsystems.

The Flask layer integrates directly with the Image Processor, which is responsible for validating uploaded images and performing preprocessing tasks such as resizing, normalization, and format conversion to ensure compatibility with the deep learning model. Additionally, the Kafka Handler class is essential for managing message exchange across different services, enabling asynchronous communication. Through Kafka, the system can handle large data streams efficiently, publishing preprocessed image tensors and receiving prediction responses in real time, thereby enhancing scalability.

The CNN Model component represents the core of the classification functionality. It includes methods for loading a trained model and making disease predictions through Predict(). Once the model generates a prediction, this information is routed to the Disease Classification module, which interprets the model's output, classifying the skin condition and calculating the likelihood or confidence level of the prediction.

.

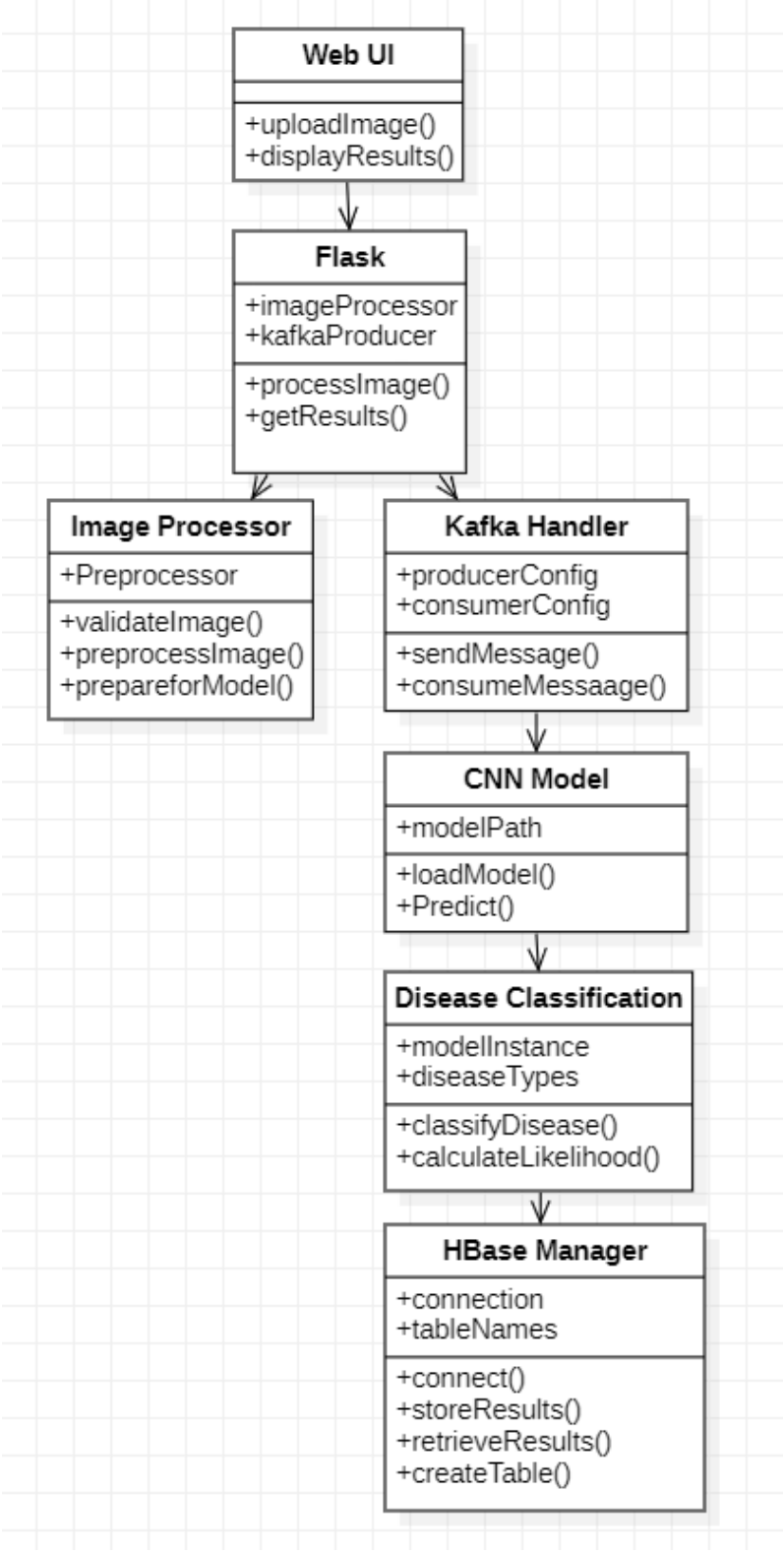


Fig 3.3: Class Diagram

Sequence Diagram

A sequence diagram is a type of UML diagram used to visualize the interactions between objects in a system over time. **Fig. 3.4** depicts the sequence diagram for our project. This sequence diagram illustrates the workflow of an image classification system. The process begins when the User uploads an image through the Web UI. The Web UI submits the image to theFlask backend, which queues the image request via the KafkaProducer. The KafkaConsumer then processes the image request and passes it to the CNN Model for analysis. The image is preprocessed, the model is applied, and a prediction is generated. The results are stored in the HBase database, and the storage confirmation is sent back to Flask. Finally, the results are returned to the Web UI, where the prediction is displayed to the User.

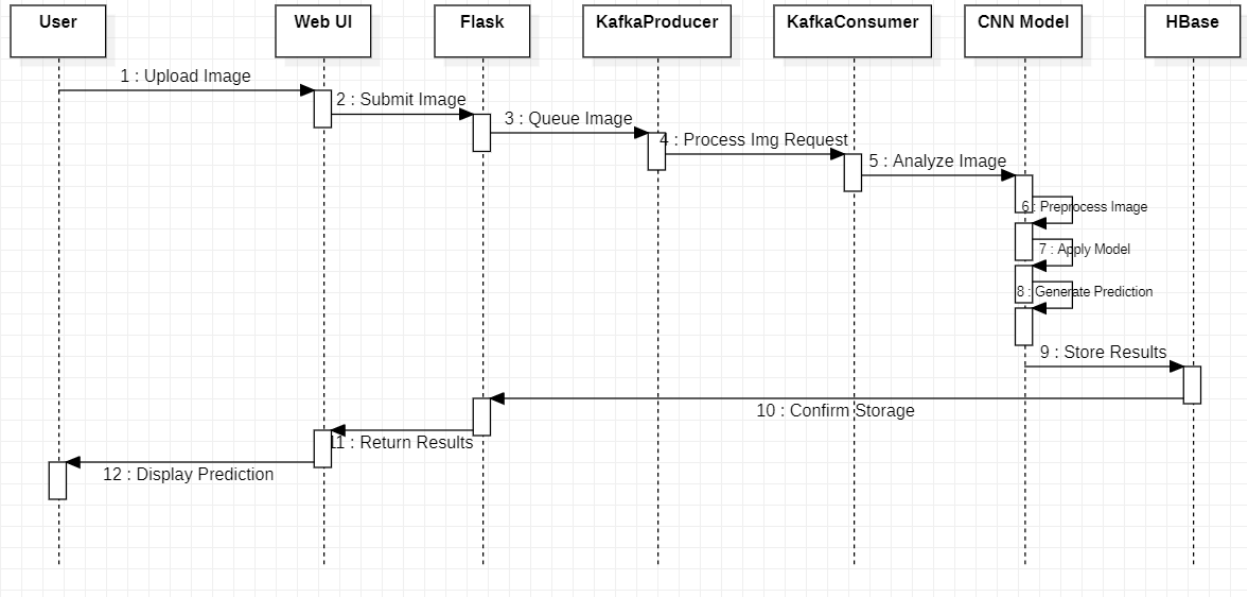


Fig. 3.4: Sequence Diagram

This project presents a comprehensive and scalable solution for skin disease classification, leveraging Apache Kafka, HBase, and a CNN model to process and analyze dermatological images in real time. By systematically handling image uploads, streaming, storage, and classification, the architecture provides accurate diagnostic insights, which are essential for timely and effective healthcare responses. The use of Kafka ensures efficient data streaming, while HBase enables secure, scalable storage of images and results, supporting the large data volumes typical in medical applications. The CNN model, trained on an extensive dataset, is optimized to capture intricate visual patterns associated with different skin conditions, achieving high accuracy.

The project's data pipeline transforms raw image data into valuable diagnostic insights, allowing healthcare providers and users to respond promptly to potential skin issues. Future enhancements could involve expanding the model to detect a broader range of skin conditions or integrating additional data sources, such as patient history, to improve diagnostic accuracy further. Overall, this project demonstrates a robust and practical framework for skin disease classification that promotes accessible, data-driven healthcare through real-time image analysis and classification.

CHAPTER-4

IMPLEMENTATION EARLY DETECTION AND CLASSIFICATION OF SKIN DISEASES

In the realm of healthcare, early and accurate detection of skin diseases is essential for improving patient outcomes and minimizing long-term impacts. With advancements in image processing and deep learning, automated skin disease classification systems have the potential to assist healthcare providers in diagnosing conditions more efficiently and accurately. This project presents a scalable framework for real-time skin disease detection using a combination of powerful tools and technologies tailored to handle large image datasets with high precision.

The system leverages Python for backend development and CNN model training, while HTML, CSS, and JavaScript enable an intuitive frontend interface for users. Key tools include TensorFlow/Keras for building and training the CNN model, OpenCV for pre-processing image data, Apache Kafka for real-time data streaming, and HBase for secure, scalable data storage. Together, these tools form an end-to-end pipeline that facilitates image ingestion, processing, storage, and classification.

The CNN model, trained on a labeled dataset of skin disease images, uses advanced deep learning techniques to classify images accurately. Through forward passes, the model processes each input image, extracting complex features across multiple CNN layers. Loss calculation against true labels enables model optimization, ensuring high accuracy across various skin conditions. This approach allows healthcare providers and users to access reliable diagnostic insights through a user-friendly interface, supporting prompt intervention and personalized care.

4.1 System Design

The system architecture comprises five integrated modules that collectively deliver a streamlined skin disease classification experience. The Front-End Module is a web-based user interface, developed using HTML, CSS, and JavaScript, which allows users to upload images and view the classification results seamlessly. The Back-End Module is managed by a Flask server

that processes HTTP requests, handles image preprocessing, and interfaces with the CNN model, coordinating data flow between the front end, the model, and storage. The CNN Module serves as the classification core, utilizing a Convolutional Neural Network (CNN) built and trained with TensorFlow or PyTorch. This model is optimized on a comprehensive dataset for high-accuracy skin disease classification. To ensure real-time performance, the Streaming Module incorporates Apache Kafka for efficient data transmission, enabling rapid processing of user-uploaded images by the model. Finally, the Storage Module employs HBase as a secure and scalable repository for storing processed images and results, allowing for long-term data retention and supporting ongoing analysis and model improvements. Together, these modules establish a robust pipeline for delivering timely, accurate, and accessible skin disease classification.

4.2 Coding and Tools

1. Python for backend development and model training
2. HTML, CSS, and JavaScript for frontend development
3. TensorFlow/Keras for deep learning
4. OpenCV for image processing
5. Kafka for data streaming
6. HBase for data storage

Model Training:

The CNN model trains on a labeled data skin disease image. In case if the number of examples on any type of skin diseases is considerable. Training follows

Forward Pass: passes an image data through different levels of CNN layers generates their predictions.

Loss Calculation: The class probabilities predicted by the model is used to calculate the loss or error against the true labels.

After training and validating the CNN model, which can be used as it is in the application. Input Image: This method involves uploading an image that the user wishes to transfer through the web interface so that it can be pre-processed to fit the standard format of the input needed by the CNN model. It feeds the preprocessed image through the trained CNN, which returns the probability values for all the categories of skin diseases.

Classification Result: It classifies the image among the six skin diseases with regard to the maximum probability, and the result is returned to the user in real time.

Confusion matrix **[Fig:4.1]** is used to find the model accuracy. For this skin disease classification model, the confusion matrix can provide insight into the model's effectiveness by showing how well it classifies each disease category.

Since this skin disease classification is a multiclass problem [6 classes] the confusion matrix is a square matrix where each row represents the actual class (true condition) of the images in the test set and each column represents the predicted class (model's output) for those images.

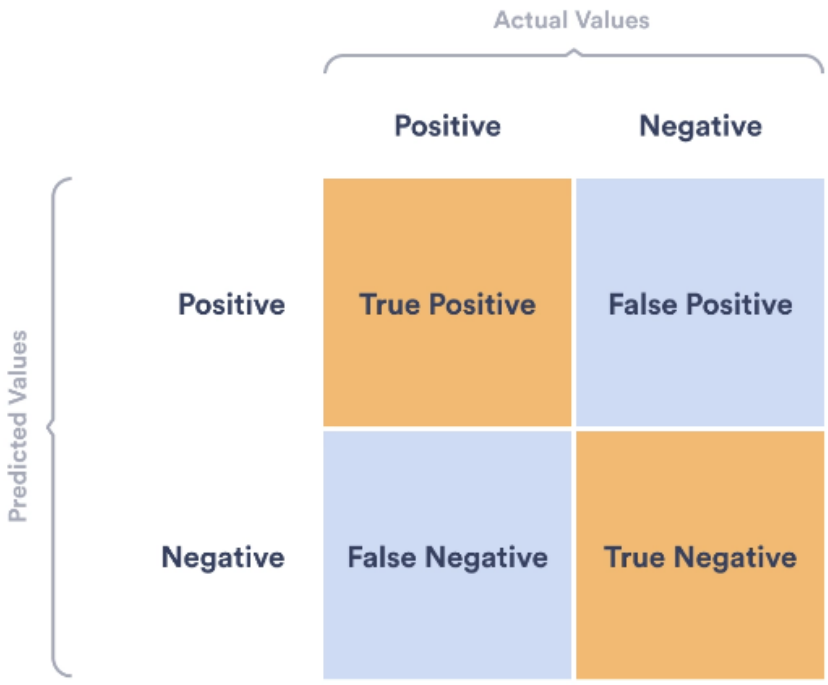


Fig: 4.1 Confusion Matrix

The React-based frontend for a Skin Disease Detection application provides an intuitive and engaging user interface, allowing users to upload images of skin lesions and receive instant AI-generated predictions. The app is structured to simplify the process, enabling users to upload an image, submit it for analysis, and view the predicted skin disease along with a confidence score.

The core functionality revolves around a seamless image upload and processing experience. Users can select files in common formats (.png, .jpg, .jpeg) through a clear and accessible file input interface. Once an image is chosen, it is immediately displayed on the screen, preparing it for the next steps. Upon clicking the "Predict" button, the app sends the image to the backend server in a POST request. This interaction is handled by an asynchronous function to ensure a smooth and responsive user experience, with the application awaiting the backend's response. Once the prediction is processed and returned, the app displays the identified skin disease and its corresponding likelihood score, prominently positioned on the right side of the page for easy viewing. In case of any errors during the upload or prediction process, a clear and helpful error message guides users, ensuring they understand what went wrong and can proceed with confidence.

The user interface is thoughtfully divided into two sections. On the left side, the application includes branding elements and a concise description of the functionality, along with an important disclaimer indicating that the predictions are AI-generated and should not replace professional medical consultation. The right section contains the file upload and prediction display, showcasing the uploaded image and providing feedback on the prediction.

The design focuses on enhancing the user experience with real-time feedback and a responsive layout. Previous results are automatically cleared when a new image is uploaded, providing users with a fresh start each time. The layout is styled using Bootstrap classes (container-fluid, row, col-lg-6) to ensure responsiveness and optimize the app's performance across a variety of devices, from desktops to mobile screens, making it widely accessible. This careful attention to layout and user feedback contributes to a polished, user-centered experience for individuals seeking instant skin disease predictions.

CSS Implementation for Skin Disease Detection Frontend

The CSS styling for the Skin Disease Detection web application is crafted to create a visually appealing, user-friendly, and professional interface. A full-screen background image gives the app a polished and engaging look, with the layout centered to keep the content prominent across different screen sizes. Uploaded images are displayed within a rounded container with a subtle brown border, creating a clear and attractive frame, while the image itself is enlarged to 150% for enhanced clarity. The Upload and Predict buttons are prominently positioned at the bottom of the page, styled with rounded corners, shadow effects, and interactive color changes on hover for a modern, intuitive feel.

Key text elements, such as the title, likelihood score, and disease prediction, use custom fonts to ensure readability and visual appeal, along with thoughtfully placed notes and instructions to guide the user. The application’s responsive design ensures that these styling features are maintained across devices, from desktop to mobile, providing a consistently seamless and engaging user experience.

Inserting data into H-Base from local

Step 1: Import Required Libraries

To begin, import the necessary libraries for the script's functionality. The `os` module is used for file and directory operations, allowing navigation and checking of image files. The `base64` module is required to encode binary image data into text format for compatibility with HBase storage. Finally, the `happybase` library is used to connect to HBase and manage data insertion.

Step 2: Establish HBase Connection

Establish a connection to the HBase server using `happybase`. Connect to `localhost` by default or specify a different server address if needed. Set the connection port to `9090` to facilitate communication with the HBase server, ensuring a stable connection for data insertion.

Step 3: Define Folders and Corresponding HBase Tables

Create a dictionary that maps each disease category folder to a unique HBase table (e.g., "AtopicDermatitis" maps to `dis0`, "Melanoma" maps to `dis5`, etc.). Also, define the column family structure, particularly `image_data`, where each encoded image will be stored. This structure ensures organized data storage and easier access for each disease category.

Step 4: Set Base Directory for Image Folders

Define the base directory that contains all disease image folders. This directory will be used as the starting point for accessing each folder in the dictionary, allowing the code to retrieve image files within each disease folder and map them to the corresponding HBase table.

Step 5: Loop Through Folders and Create Tables if Needed

For each `folder_name` and `table_name` pair in the dictionary, check if the table already exists in HBase. If it does, print a confirmation message to notify the user that the table is available. If the table does not exist, create it with the `image_data` column family and print a success message to confirm the new table setup. This step avoids unnecessary table duplication and ensures a structured setup for each disease category.

Step 6: Access the Table and Insert Images

Access the specified image folder for each disease category by building the folder path using the base directory and folder name. Retrieve the corresponding HBase table and prepare it for image insertion. This setup aligns each image directory with its corresponding HBase table for efficient data insertion.

Step 7: Loop Through Images in Each Folder and Insert Data

Within each folder, iterate through the files to find images. Verify if each file is an image by checking its file extension (e.g., `png`, `jpg`, etc.). If the file is confirmed as an image, open it in binary mode, read the content, and encode it in base64 format to convert it into a text string. Generate a unique `row_key` for each image based on the filename (excluding the extension), then insert the encoded image into the HBase table using the `put` method, storing it in the `image_data:image` column. This ensures that each image is accurately stored and easily retrievable.

Step 8: Close the HBase Connection

Once all tables and image data have been processed, close the HBase connection to free up resources and maintain best practices. This step completes the data insertion process and ensures the HBase environment is ready for any future operations.

Code for FLASK_KAFKA PRODUCER:

Step 1: Import Required Libraries

To set up this Flask application, import various libraries essential for image processing, model loading, Kafka messaging, and CORS support. The `os` and `secure_filename` from `werkzeug` handle file and directory operations, while `PIL.Image` and `numpy` (`np`) are used to process image data for prediction. `Flask` and `flask_cors.CORS` enable the API setup and CORS policy handling, allowing cross-origin requests. `tensorflow.keras` loads the pre-trained CNN model, and `kafka` provides Kafka functionality for real-time data streaming. `json` manages JSON serialization for message formatting.

Step 2: Configure Flask Application and Define Upload Directory

Initialize a Flask app instance and enable CORS with `CORS(app)` to allow API access from other origins. Configure the file upload directory by setting `UPLOAD_FOLDER` in `app.config` to store temporarily uploaded files.

Step 3: Load CNN Model and Define Class Names

Load the pre-trained CNN model for skin disease classification using `load_model` and specify the model path. Define the `class_names` list containing skin disease categories that the model can predict. This list provides an index-based reference to map prediction results to human-readable labels.

Step 4: Setup Kafka Producer

Create a Kafka producer instance to send image prediction data to a Kafka server. Configure the Kafka producer with `bootstrap_servers` to `localhost:29092` and set the `value_serializer` to JSON-encode messages. This setup enables the application to send structured prediction data to Kafka in real-time.

Step 5: Define Helper Functions

Define essential helper functions: allowed_file(filename): Checks if the uploaded file has an allowed extension (PNG, JPG, or JPEG), ensuring only valid image files are processed. prepare_image(image): Converts an uploaded image to RGB format, resizes it to `(128, 128)` pixels to match the CNN model input, and reshapes it into a suitable array format.

Step 6: Define Homepage Route

Define a simple route `/` to serve as the homepage, which returns an HTML page with instructions. This page informs users about the purpose of the API and how to use it.

Step 7: Define Prediction Route

Set up the `/predict` route with the `POST` method to handle image prediction requests.

Step 8: Define Kafka Message Sending Function

Define `send_image_to_kafka()` to handle the sending of image prediction data to a Kafka topic. This function performs the following:

1. Topic Creation: Ensures the topic exists by calling `create_kafka_topic()`. If it doesn't exist, the topic is created.

2. Base64 Encode Image: Converts image content to base64 to make it compatible with JSON format and Kafka.

3. Prepare Message: Builds a structured message containing `image_data`, `predicted_class`, and `likelihood`.

4. Send Message: Sends the message to the Kafka topic and flushes the producer to ensure delivery.

Step 9: Define Kafka Topic Creation Function

Define `create_kafka_topic()` to dynamically create a Kafka topic if it does not already exist. Using `KafkaAdminClient`, this function:

1. Setup Topic Parameters: Specifies the new topic name, number of partitions, and replication factor.

2. Create Topic: Attempts to create the topic using `create_topics()`. If the topic already exists, a message is logged. Any other errors are caught and printed.

3. Close Admin Client: Ensures the Kafka admin client connection is closed after topic creation.

Step 10: Start Flask Application

In the main block, call `app.run()` with `debug=True` to start the Flask application in debug mode, enabling error logging and live reloading for development.

Code for Kafka_consumer :

Step 1: Import Required Libraries

Import necessary libraries to support Kafka and HBase interactions, JSON handling, UUID for unique keys, and logging for status messages. This includes KafkaConsumer for consuming messages from Kafka, json for parsing JSON data, happybase for HBase operations, and logging for tracking events.

Step 2: Configure Logging and Define Custom Exception

Set up a logging configuration to capture and display formatted log messages, making it easier to monitor consumer activity and troubleshoot. Define a custom exception, HBaseConnectionError, to handle specific errors related to HBase connection issues.

Step 3: Initialize DiseasePredictionConsumer Class

This class is responsible for managing connections to Kafka and HBase. It initializes these connections and includes retry logic to ensure stable operation. The constructor takes Kafka and HBase configurations as parameters, stores them, and calls _initialize_connections() to establish initial connections.

Step 4: Set Up Kafka and HBase Connections with Retry Logic

This step involves two methods: _init_kafka_consumer() and _init_hbase_connection(), both of which include retry logic to attempt reconnections if initial attempts fail.

Method _init_kafka_consumer(): Connects to Kafka with retry logic. Logs each attempt and raises an exception if the connection fails after a specified number of retries.

Method _init_hbase_connection(): Connects to HBase, also with retry logic. Attempts reconnections upon failure and raises HBaseConnectionError if unsuccessful after the maximum number of retries.

Step 5: Ensure the Required HBase Table Exists

The _ensure_table_exists() method checks if a table for the specific disease exists in HBase. It dynamically creates the table with two column families (info and metrics) if it does not already exist. Logs are generated to confirm table creation or to indicate that the table is already present.

Step 6: Store Message Data in HBase

The _store_in_hbase() method handles the storage of data in HBase with retry logic. It defines a row_key using a combination of timestamp and a unique ID, and the data is stored in a dynamically selected table. If storage fails, the connection to HBase is reinitialized and the process is retried up to the specified limit.

Step 7: Start Consuming Messages from Kafka

The start_consuming() method initiates the continuous process of consuming messages from Kafka. For each message, extracts data like predicted_class to define the table name. Prepares a row_key and constructs a dictionary with disease information and metrics. Calls _store_in_hbase() to save the message in HBase. Logs success messages and handles any storage-related errors.

Step 8: Gracefully Handle Application Shutdown

The close() method safely closes connections to Kafka and HBase during application shutdown. This ensures that resources are freed and that the application shuts down smoothly. Logs indicate the status of each connection closure.

Step 9: Start the Consumer in the Main Block

In the main section of the code, a configuration dictionary for Kafka and HBase settings is created. A DiseasePredictionConsumer instance is initialized with these settings, and start_consuming() is called to begin processing messages. Errors are captured and logged in case of failure.

Verification code for Hbase table creation:

Step 1: Import Necessary Libraries

In this step, we import the essential libraries required for the task. happybase is used to manage the connection to HBase and interact with its data. The base64 library is utilized to decode base64-encoded image data stored in HBase. PIL.Image from the Python Imaging Library (PIL) is used to handle and display the image data, while BytesIO from the io module allows for in-memory manipulation of binary image data. These libraries collectively enable us to retrieve, decode, and display images from HBase efficiently.

Step 2: Establish Connection with HBase Server

In this step, we establish a connection to the HBase server using the happybase.Connection class. The connection is made to localhost at port 9090, assuming the HBase server is running on the local machine. If your HBase server is hosted on a different machine, you would replace 'localhost' with the appropriate IP address or hostname of the server. The connection is essential for querying the HBase table that stores the image data.

Step 3: Define and Access the Table

Here we defined (tinea_ringworm_candidiasis_and_other_fungal_infections_pred) from which we want to retrieve image data. The connection.table(table_name) command accesses the specified table in HBase. The table holds the rows with image data stored in the info:image_path column. By accessing the table, we can scan it and retrieve the images stored in the rows for further processing.

Step 4: Retrieve Row Keys for Images

In this step, we scan the HBase table to retrieve the rows that contain image data. Using the table.scan() method, we specify a limit of 10 rows to retrieve, ensuring that only the first 10 rows (images) are fetched from the table. This step is critical for managing the number of rows processed, especially when dealing with large datasets, as it ensures that we don't retrieve too many rows at once, which could be inefficient or overwhelming.

Step 5: Initialize Counter for Displayed Images

A counter is initialized to keep track of how many images are displayed. The variable image_count is set to 0 at the beginning, and a max_images limit of 10 is set to control how many images will be shown. This counter ensures that the script doesn't attempt to display more than the desired number of images. Once the counter reaches the max_images limit, the loop will stop displaying images, preventing unnecessary processing.

Step 6: Loop Through Rows and Process Images

In this step, we loop through the rows retrieved from the HBase table and process each one to extract and display the images. For each row, the image is stored in the info:image_path column

family, and we attempt to retrieve this base64-encoded image data. If the image exists, it is decoded from its base64 format and opened using PIL's Image.open() function. The image is then displayed with the row key as its title. Error handling is included in this step to ensure that if there's any issue with decoding or displaying the image, it will be logged and the process will continue with the next image. Additionally, if the number of displayed images exceeds the max_images limit, the loop breaks, ensuring the process remains within the desired range.

Step 7: Close the HBase Connection

Once the images have been processed and displayed, it is important to close the connection to HBase. This step ensures that all resources are released and that no unnecessary connections remain open. Closing the connection is a good practice to avoid resource leaks and ensure the program operates efficiently. This is done using the connection.close() method, which properly terminates the connection to the HBase server.

Step 8: Print Summary of Displayed Images

Finally, after all the images are displayed, a summary message is printed to inform the user of how many images were successfully displayed. This message uses the image_count variable, which was incremented each time an image was displayed, to provide feedback on the process. This step is useful for tracking the script's execution and ensuring that the expected number of images were processed and shown.

CNN Model:

Step 1: Import Necessary Libraries

In this step we set up the environment by disabling ONEDNN optimizations with os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0' to avoid compatibility issues. We then import TensorFlow (import tensorflow as tf) for building the CNN model, Matplotlib and Seaborn (import matplotlib.pyplot as plt, import seaborn as sns) for visualization, and Pandas (import pandas as pd) for data handling, preparing the environment for model building and analysis.

Step 2: Data Preparation and Augmentation

We prepare the training and validation datasets by loading them from the specified directory using `tf.keras.utils.image_dataset_from_directory`. This function automatically infers the labels based on the directory structure. The training set consists of 80% of the data, and the validation set contains 20%, as specified by the `validation_split` parameter. Both datasets are resized to a uniform shape of (128, 128) using `image_size` and are batched in groups of 32 images for efficient training. The `label_mode="categorical"` ensures that labels are one-hot encoded, suitable for classification tasks. The training set is further augmented by applying random transformations such as horizontal flipping, brightness adjustments, and contrast modifications through the `augment()` function, enhancing model robustness and generalization by artificially increasing the diversity of training data.

Step 3: Building the CNN Model

In this step, we construct a Convolutional Neural Network (CNN) to process the input image data for classification tasks. The model starts with an Input Layer that defines the shape of the input images as 128x128 pixels with 3 color channels (RGB). The architecture follows a Sequential model structure, where each layer is added one after the other. The core of this model is a series of Convolutional Blocks designed to extract increasingly complex features from the images. The first convolutional block consists of two Conv2D layers, each using a 3x3 kernel with 32 filters, followed by a MaxPooling layer that reduces the spatial dimensions by a factor of two. This pattern is repeated for subsequent blocks, where the number of filters increases as the network deepens: 64 filters in the second block, 128 in the third, 256 in the fourth, and 512 in the fifth block. These convolutional layers are followed by ReLU activation functions, which introduce non-linearity, enabling the model to learn more complex patterns in the data.

After the convolutional layers, a Dropout layer with a 0.25 dropout rate is added to mitigate overfitting by randomly setting a fraction of the input units to zero during training. Then, the Flatten layer reshapes the feature maps into a 1D vector, which is fed into a Dense layer with 1500 units. This fully connected layer allows the model to make sense of the features extracted by the convolutional blocks and learn higher-level representations. Another Dropout layer with a 0.4

dropout rate is applied before the final Output Layer, which consists of 6 units (corresponding to 6 classes in this classification task) and uses the softmax activation function. This ensures that the model outputs a probability distribution over the 6 classes.

Finally, the model is compiled using the Adam optimizer with a learning rate of 0.0001, chosen for its adaptive learning rate properties, and categorical crossentropy as the loss function, which is appropriate for multi-class classification problems. The model's performance is evaluated based on the accuracy metric, which tracks how often the predicted class matches the true class. This CNN architecture is designed to efficiently learn and classify images from the dataset into their respective categories.

Step 4: Training the Model

In this step, the model is trained using the fit() method, where the training data (`train_set`) is passed in along with the validation data (`validation_set`). The training process runs for 20 epochs, allowing the model to learn patterns from the training data while periodically validating its performance on the validation set. The `epochs=20` argument ensures that the model undergoes 20 full passes through the dataset. This step tracks the model's performance on both training and validation datasets, providing insights into how well it generalizes.

Step 5: Evaluating the Model

After training, the model's performance is evaluated using the evaluate() method on the validation set to calculate the final validation loss and validation accuracy. This step helps determine how well the model has learned to classify new, unseen data. The model is then saved using `cnn.save('cleansed_disease_model.keras')`, storing the trained model for future use or deployment.

Step 6: Training and Validation Set Accuracy

In this step, the model's performance is further evaluated on both the training and validation datasets using the evaluate() method. This step provides detailed insights into the accuracy of the model during both training and validation. The model is saved again after 20 epochs using a new file name `'cleansed_20_epoch.keras'` to ensure the latest model is stored.

Step 7: Visualizing Training and Validation Accuracy

The training history, obtained from the `training_history.history`, is used to plot a graph that compares the training accuracy and validation accuracy over 10 epochs. The `matplotlib` library is used to plot these accuracy values, providing a visual representation of how the model performs over time. The plot helps to assess if the model is overfitting or if the training and validation accuracies are improving similarly.

Step 8: Confusion Matrix Visualization

To better understand how the model is performing for each class, a confusion matrix is computed. This matrix compares the true labels (`y_true`) with the predicted labels (`y_pred`) to show the number of correct and incorrect predictions for each class. A heatmap of the confusion matrix is then plotted using `seaborn` to provide a clear and intuitive visual representation of the classification performance.

Step 9: Calculating Additional Evaluation Metrics

Finally, additional evaluation metrics such as accuracy, precision, recall, F1-score, and Cohen's Kappa score are calculated using the predictions from the validation set. These metrics provide a deeper understanding of the model's classification performance, including how well it balances precision and recall, and its overall agreement with the true labels. Each metric is printed to assess the model's strengths and weaknesses in the classification task.

4.3 Process Flow

The process flow for the skin disease detection application involves a seamless interaction between the user, the web interface, the backend server, and the machine learning model. Below is an expanded explanation of each step:

User uploads an image: The process begins with the user selecting and uploading an image of a skin lesion through the web interface. This image typically needs to be a clear, zoomed-in photo of the affected area to allow for accurate analysis.

The image is preprocessed and sent to the backend server: Once the image is uploaded, it undergoes preprocessing. This step may include resizing, normalization, or adjusting image

contrast to make it suitable for input into the CNN model. This preprocessing ensures that the image is in the correct format, and any noise or unnecessary information is removed. The preprocessed image is then sent to the backend server, where the model is hosted.

The server processes the image and sends it to the CNN model for classification: On the backend, the server receives the image and feeds it into the CNN model, which has been trained on a large dataset of skin lesion images. The server ensures efficient handling of the image and initiates the classification process by passing the image through the trained layers of the CNN.

The model predicts the most likely skin disease category: The CNN model processes the image and generates predictions based on its learned patterns from the training data. It outputs a probability score for each possible skin disease category, and the category with the highest probability is selected as the most likely diagnosis.

The prediction result is displayed to the user on the web interface: The server sends the prediction result back to the frontend. The result is displayed to the user in an easy-to-read format, showing the predicted skin condition along with a confidence score (likelihood percentage). This immediate feedback allows users to get quick insight into their skin condition.

The image and prediction are stored in the HBase database for future analysis: After the prediction is made, both the uploaded image and its corresponding prediction are stored in the HBase database. Storing this data allows for future analysis, such as tracking user history, model performance evaluation, or further improvements to the model by retraining it with new data. This step ensures the system can learn and improve over time.

4.4 Challenges and Obstacles

1. **Data Quality:** The accuracy of any machine learning model, especially in medical image classification, heavily depends on the quality and diversity of the training dataset. In this case, the training data needs to include a wide variety of skin types, lesion types, lighting conditions, and image qualities to ensure the model generalizes well across different users. If the dataset is biased or lacks sufficient diversity, the model may perform poorly on real-world data, leading to inaccurate predictions and reduced reliability.

2. **Model Complexity:** Building a CNN that is both accurate and efficient can be a difficult balance. A highly complex model with many layers and parameters may achieve higher accuracy, but it may also require substantial computational resources, leading to slower predictions and higher costs. On the other hand, a simpler model may be faster but less accurate. Finding the right balance between model complexity and resource efficiency is crucial, especially when deploying the model in real-time web applications. Ensuring scalability while maintaining performance is another consideration as user numbers grow.

3. **User Experience:** The success of the application also depends on its user interface. A user-friendly and intuitive design is essential for encouraging widespread adoption, especially for non-technical users. The process of uploading images and receiving predictions should be simple, with clear instructions and minimal technical jargon. Additionally, providing informative feedback to the user, such as explanations of the predicted skin conditions and confidence scores, helps build trust in the application. Any delays in response or confusing layouts could lead to a poor user experience, decreasing the likelihood of users relying on the application regularly.

This project successfully implements a robust and scalable skin disease classification system, leveraging advanced technologies such as CNNs, Apache Kafka, and HBase to process and analyze dermatological images in real time. By combining deep learning with efficient data streaming and storage solutions, the system is capable of delivering accurate and timely diagnostic insights, which are crucial for early detection and effective healthcare intervention. The use of Python, TensorFlow/Keras, and OpenCV ensures the backend is capable of handling complex image processing tasks, while the frontend provides a seamless user experience for easy interaction.

The model's ability to classify skin diseases with high accuracy, coupled with real-time data processing, makes it a valuable tool for both healthcare professionals and users seeking quick diagnostic feedback. Future work may focus on expanding the model's capabilities to detect a wider range of skin conditions or integrating additional data sources for more comprehensive analyses. Ultimately, this project demonstrates the potential of deep learning and big data technologies in transforming healthcare delivery, making skin disease detection more accessible, efficient, and reliable.

CHAPTER-5

RESULTS AND DISCUSSION OF REAL-TIME DERMATOLOGICAL IMAGE ANALYSIS FOR EARLY SKIN DISEASE DIAGNOSIS

5.1 Output:

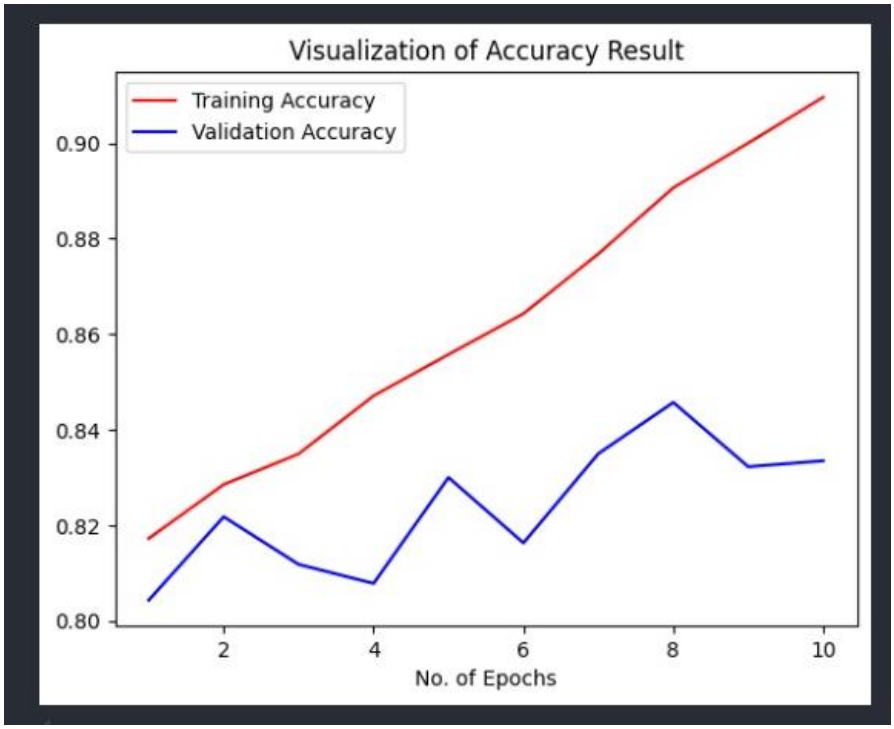


Fig 5.1: Result accuracy

The accuracy of fig 5.1 plot shows that the model demonstrates effective learning over time, with training accuracy steadily improving across all 10 epochs. This consistent rise highlights the model's ability to capture patterns in the training data, reaching over 90% accuracy by the final epoch. Additionally, the validation accuracy maintains a solid baseline around 82-84%, indicating that the model is making reasonable predictions on unseen data. This stable validation performance suggests that the model has a good foundation for generalization. With the high learning capacity observed in the training accuracy, there is promising potential for further refinement to improve validation performance, which could lead to even better generalization on new data.

In Figure 5.2, the Kafka consumer terminal displays the base64-encoded version of an image being consumed from a Kafka topic, such as `new_topic`. This encoded format is generated by converting the raw image content into a base64 string, which is then decoded to UTF-8 for display in the terminal. This encoding approach allows Kafka to handle and transmit binary image data in a text-based format, ensuring compatibility with Kafka's messaging system. The encoded image can be easily decoded back to its original form when needed, allowing for efficient and seamless transmission of multimedia content within the Kafka pipeline.

Fig 5.2: Kafka consuming topics

In fig 5.3 the confusion matrix provides a comprehensive view of the CNN model's performance in predicting six classes of skin diseases. Overall, the model demonstrates strong classification abilities, with notable accuracy in identifying several classes, particularly class 2. This class had the highest number of correct predictions, with 1,419 true positives, reflecting the model's strong ability to recognize this specific skin condition. In other classes, such as class 0 and class 5, the model also performed fairly well, with 577 and 295 correct classifications, respectively. However, there is some level of misclassification between closely related classes, such as between class 0 and class 1, and between class 4 and class 5. These confusions suggest some overlap in the model's perception of these conditions, possibly due to visual similarities between the diseases. the model shows promise with a significant number of correct predictions across most classes. The concentration of true positives, especially in the major classes, indicates that the CNN is effective overall in detecting skin conditions. The focus now could be on refining

the model to reduce misclassifications in certain challenging class pairs, further enhancing its predictive power

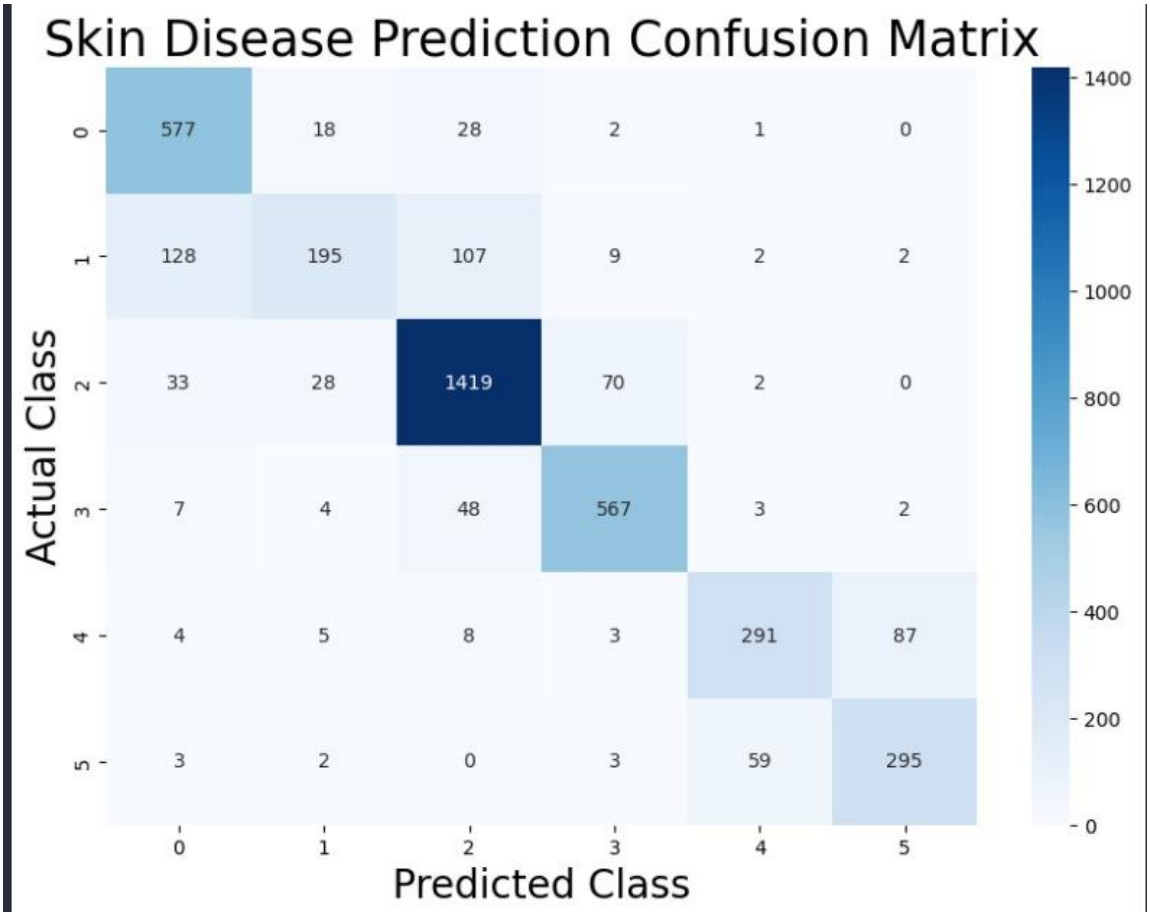


Fig 5.3: Confusion Matrix

In **Fig 5.4** the training history shows a positive trend in both accuracy improvement and loss reduction over the 10 epochs. Starting with an accuracy of 47.11% in epoch 1, the model steadily improved, reaching 79.98% by epoch 10. Simultaneously, the training loss decreased significantly from 1.3460 to 0.5065, indicating that the model became more accurate and confident in its predictions. The validation accuracy also improved, beginning at 64.36% in the first epoch and increasing to 81.16% by the final epoch, while the validation loss decreased from 0.8615 to 0.4889.The training was likely stopped after 10 epochs because the model's performance had reached a high accuracy, and further training might not yield significant improvement or could risk overfitting. The validation loss and accuracy had also stabilized, indicating the model had reached a good balance between fitting the training data and generalizing well to the validation data.

```
training_history = cnn.fit(x=train_set,validation_data=validation_set,epochs=10)
```

```
Epoch 1/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 118s 219ms/step - accuracy: 0.4711 - loss: 1.3460 - val_accuracy: 0.6436 - val_loss: 0.8615
Epoch 2/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 106s 211ms/step - accuracy: 0.6496 - loss: 0.8718 - val_accuracy: 0.7183 - val_loss: 0.7366
Epoch 3/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 105s 210ms/step - accuracy: 0.6922 - loss: 0.7731 - val_accuracy: 0.6757 - val_loss: 0.7547
Epoch 4/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 139s 204ms/step - accuracy: 0.7167 - loss: 0.7084 - val_accuracy: 0.7460 - val_loss: 0.6250
Epoch 5/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 145s 210ms/step - accuracy: 0.7447 - loss: 0.6516 - val_accuracy: 0.7732 - val_loss: 0.5876
Epoch 6/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 103s 205ms/step - accuracy: 0.7638 - loss: 0.6108 - val_accuracy: 0.7597 - val_loss: 0.5903
Epoch 7/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 148s 216ms/step - accuracy: 0.7721 - loss: 0.5944 - val_accuracy: 0.7764 - val_loss: 0.5742
Epoch 8/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 106s 210ms/step - accuracy: 0.7851 - loss: 0.5489 - val_accuracy: 0.7846 - val_loss: 0.5374
Epoch 9/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 138s 202ms/step - accuracy: 0.7836 - loss: 0.5447 - val_accuracy: 0.8036 - val_loss: 0.4954
Epoch 10/10
502/502 ━━━━━━━━━━━━━━━━━━━━ 110s 219ms/step - accuracy: 0.7998 - loss: 0.5065 - val_accuracy: 0.8116 - val_loss: 0.4889
```

Fig 5.4: Model Training

In **Fig 5.5** by using the command docker exec -it hbase hbase shell, you can access the HBase shell inside the running Docker container named hbase. Once inside the shell, the list command allows you to view all the existing tables in your HBase instance. This command will print a list of table names that are currently available. If there are no tables created yet, the result will be empty. This is a straightforward way to interact with and manage HBase tables, enabling you to perform various operations like creating, scanning, and managing tables directly from the shell environment.

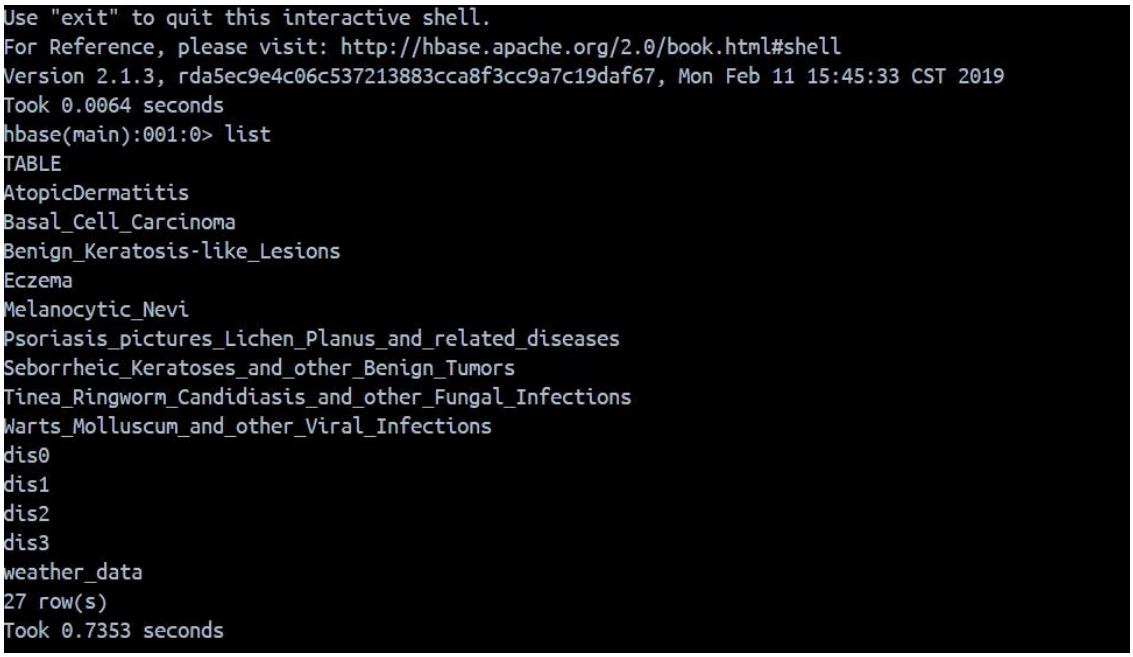


Fig 5.5: HBase Tables

In the fig 5.6 , the front-end interface of a Skin Disease Detector application is displayed. This layout allows users to upload an image of their skin condition for an instant health checkup. The design is clean and user-friendly, featuring an "Upload Image" button and a "Predict" button for submitting the image to the system. The left side of the interface provides the application's name and a note advising users that the results are AI-generated and not a substitute for professional medical advice. Once an image is uploaded, the application will display the predicted skin condition along with the likelihood percentage on the right side of the screen. However, in this particular view, no image has been uploaded, and the prediction section shows "Likelihood: 0%," indicating that no analysis has been performed yet.

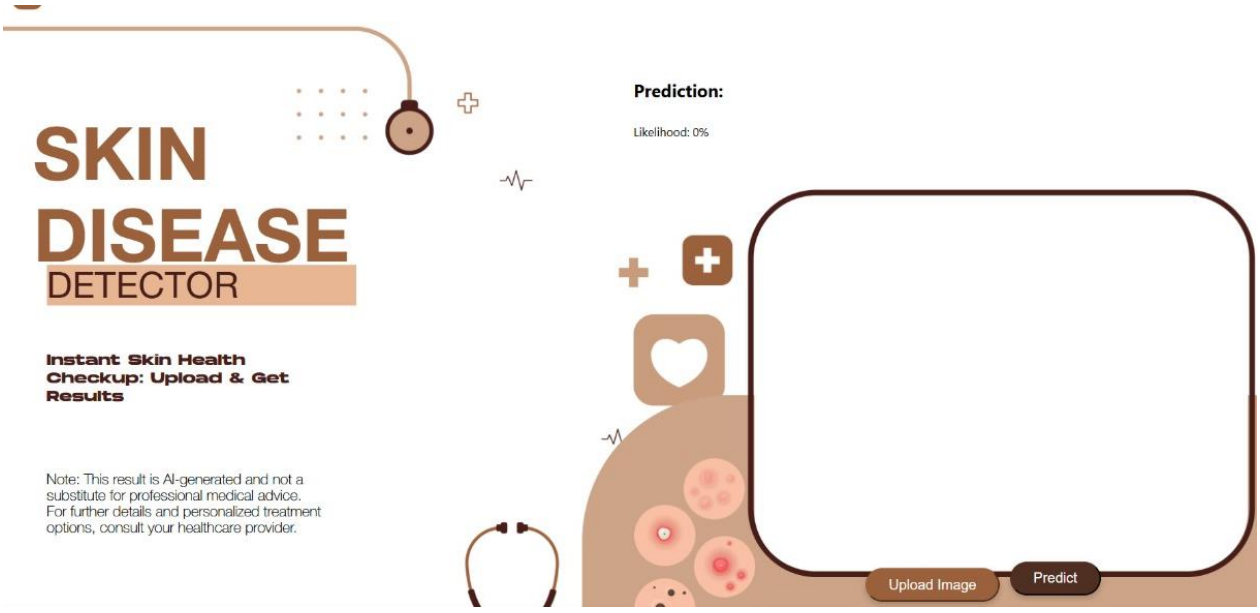


Fig 5.6: Landing Page

In the fig 5.7 , the same interface is shown as fig 5.6, but now an image has been uploaded, and the model has made a prediction. The system predicts the skin condition to be Melanocytic Nevi, with a likelihood of 91%. The uploaded image of the skin lesion is displayed in the box, providing a visual reference. The result demonstrates the model's capability of analyzing the input and providing a likely diagnosis with a confidence score, offering users quick insight into their skin condition.

Fig 5.7: Skin disease prediction

5.2 Performance Metrics:

In fig 5.8 it is displaying evaluation metrics like precision, recall, F1-score, and Cohen's Kappa from our trained CNN model, you would first use the model to predict the labels for your dataset x_train,y_train. These predictions are then compared to the true labels y_train to compute the metrics. After making predictions, you can convert the model's output (probabilities) into class labels using argmax and then use libraries like sklearn to calculate the desired metrics. Precision, in this case, is 0.8111, indicating that 81.11% of the predicted positive labels were correct. Recall is 0.7868, showing that 78.68% of the actual positives were correctly identified. The F1-score (0.7895) balances precision and recall, and Cohen's Kappa (0.7825) reflects the agreement between predicted and true labels, considering the possibility of random guesses.

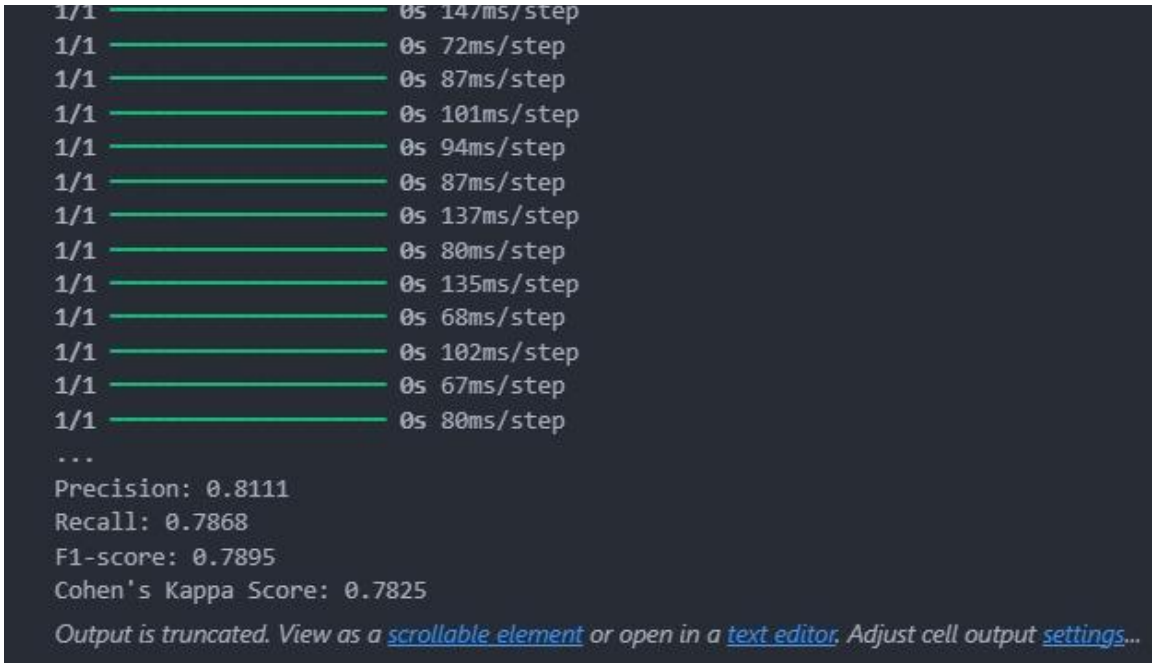


Fig: 5.8

CHAPTER-6

CONCLUSION OF REAL-TIME SKIN DISEASE IDENTIFICATION AND ClASSIFICATION USING CNN

This project succeeded in developing a robust web-based application that uses the power of Convolutional Neural Networks (CNNs) to classify common skin diseases. The application allows users to upload images of their conditions via an accessible platform. Thus, it empowers patients to take proactive steps to diagnose and manage their skin conditions at an early stage. The system will process and store data using Kafka and HBase, so the analysis will be immediate and future model improvements. Quality and diversity of the training data might affect the model's accuracy. Generalization across various skin tones and conditions might also be further studied and improved.

The dataset consists of 6 classes which are Melanocytic Nevi, Melanoma, Seborrheic Keratoses and other Benign Tumors, and Tinea (Ringworm), Candidiasis, and other Fungal Infections. The dataset is curated to offer diverse examples of each condition, reflecting a broad spectrum of visual characteristics. The dataset has been obtained from Kaggle which is of size 5.52GB.

Fig:6.1 depicts the metadata of the dataset.

Disease Type	Count	Percentage
Basal Cell Carcinoma	3,323	16.6%
Benign Keratosis	2,079	10.4%
Melanocytic Nevi	7,970	39.7%
Melanoma	3,140	15.7%
Seborrheic Keratoses	1,847	9.2%
Fungal Infections	1,702	8.5%
Total	**20,061**	**100%**

Fig: 6.1: Distribution of classes in the dataset

CHAPTER-7

Future Work Of Advanced Skin Disease Classification Using Big Data and Neural Networks

As the field of skin disease detection continues to evolve, several key improvements can be made to further enhance the capabilities and impact of the current system. Below are the potential areas for future development:

One of the most significant ways to improve the model's performance is by increasing the diversity and scope of the dataset. While the current model is designed to classify a range of common skin conditions, expanding the dataset to include rare or less common diseases would greatly increase the model's versatility. By incorporating a broader array of images, the system could become more robust and capable of handling a wide variety of dermatological conditions. This expansion would involve collecting more labeled images from reputable sources, including both well-known conditions and obscure diseases that are not commonly included in typical dermatological datasets. The inclusion of rare cases would also improve the model's ability to generalize across different skin types, colors, and environmental factors, making it more reliable in real-world clinical settings.

The model's accuracy is crucial for reliable classification, and there are several strategies to improve it. Data augmentation techniques can be implemented to artificially increase the variety of images in the training dataset by applying transformations like rotation, flipping, scaling, or color adjustments. This process would make the model more robust and help prevent overfitting to the existing dataset. Additionally, transfer learning could be leveraged to fine-tune pre-trained models like VGG16, ResNet, or EfficientNet, which have already been trained on large, diverse datasets and can be adapted to dermatological image classification. Transfer learning allows the model to benefit from the generalized features learned by these models, improving performance with fewer training examples. Lastly, ensemble methods could be employed, where multiple models are trained and their predictions are combined to produce a final output. This technique

can often result in higher accuracy by capitalizing on the strengths of different models and mitigating individual weaknesses.

For the system to have a real-world impact, especially in clinical settings, integrating the application into healthcare workflows is essential. Collaborating with healthcare providers and medical institutions could help ensure that the application meets the practical needs of dermatologists and healthcare professionals. Integrating the system into Electronic Health Records (EHR) or Picture Archiving and Communication Systems (PACS) would streamline the process of analyzing skin images and providing real-time diagnostic feedback. This clinical integration would also open up opportunities for continuous improvement, as the system could learn from new, real-world data and provide updated insights to medical professionals. Furthermore, real-time diagnostic support could aid doctors in identifying skin diseases at an early stage, ultimately leading to faster treatments, better patient outcomes, and a reduction in healthcare costs by preventing the progression of undiagnosed conditions.

By expanding the dataset, improving model accuracy through advanced techniques, and integrating the system into clinical workflows, this skin disease classification project can evolve into a more powerful, accurate, and widely applicable tool. These enhancements will enable the system to deliver even greater value to both healthcare providers and patients, helping to improve the accessibility and efficiency of dermatological care.

REFERENCES

[1] "Skin diseases image dataset," *www.kaggle.com*. https://www.kaggle.com/datasets/ismailpromus/skin-diseases-image-dataset/data

[2] "Studies on Different CNN Algorithms for Face Skin Disease Classification Based on Clinical Images | IEEE Journals & Magazine | IEEE Xplore," *ieeexplore.ieee.org*. https://ieeexplore.ieee.org/document/8720210

[3] E. Goceri, "Analysis of Deep Networks with Residual Blocks and Different Activation Functions: Classification of Skin Diseases," *2019 Ninth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Nov. 2019, doi: https://doi.org/10.1109/ipta.2019.8936083.

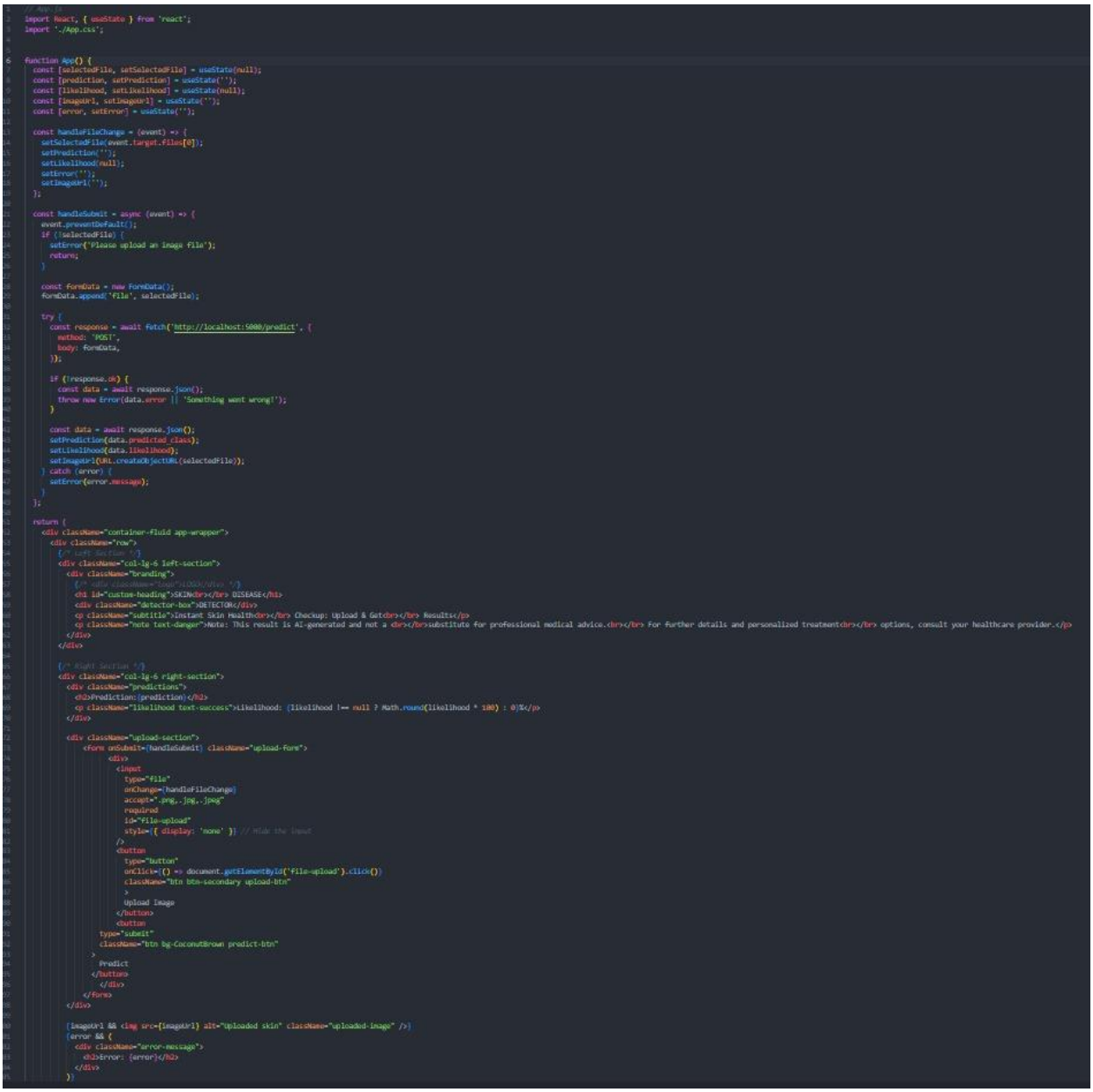
[4] G. Singh, Kalpna Guleria, and S. Sharma, "A Transfer Learning-based Pre-trained VGG16 Model for Skin Disease Classification," pp. 1–6, Dec. 2023, doi: https://doi.org/10.1109/mysurucon59703.2023.10396942.

[5] G. Cavallaro, M. Riedel, M. Richerzhagen, J. A. Benediktsson, and A. Plaza, "On Understanding Big Data Impacts in Remotely Sensed Image Classification Using Support Vector Machine Methods," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 10, pp. 4634–4646, Oct. 2015, doi: https://doi.org/10.1109/jstars.2015.2458855.

[6] N. Akmalia, P. Sihombing, and Suherman, "Skin Diseases Classification Using Local Binary Pattern and Convolutional Neural Network," *IEEE Xplore*, 2019. https://ieeexplore.ieee.org/document/8943892

Github link: https://github.com/krishnamuttevi/EARLY-DETECTION-AND-CLASSIFICATION-OF-SKIN-DISEASES
Demo link: https://youtu.be/hDQojP3JGG8?feature=shared

APPENDIX



```jsx
// App.js
import React, { useState } from 'react';
import './App.css';


function App() {
  const [selectedFile, setSelectedFile] = useState(null);
  const [prediction, setPrediction] = useState('');
  const [likelihood, setLikelihood] = useState(null);
  const [imageUrl, setImageUrl] = useState('');
  const [error, setError] = useState('');

  const handleFileChange = (event) => {
    setSelectedFile(event.target.files[0]);
    setPrediction('');
    setLikelihood(null);
    setError('');
    setImageUrl('');
  };

  const handleSubmit = async (event) => {
    event.preventDefault();
    if (!selectedFile) {
      setError('Please upload an image file');
      return;
    }

    const formData = new FormData();
    formData.append('file', selectedFile);

    try {
      const response = await fetch('http://localhost:5000/predict', {
        method: 'POST',
        body: formData,
      });

      if (!response.ok) {
        const data = await response.json();
        throw new Error(data.error || 'Something went wrong!');
      }

      const data = await response.json();
      setPrediction(data.predicted_class);
      setLikelihood(data.likelihood);
      setImageUrl(URL.createObjectURL(selectedFile));
    } catch (error) {
      setError(error.message);
    }
  };

  return (
    <div className="container-fluid app-wrapper">
      <div className="row">
        {/* Left Section */}
        <div className="col-lg-6 left-section">
          <div className="branding">
            {/* <div></div> */}
            <h1 id="custom-heading">SKIN<br></br> DISEASE</h1>
            <div id="detector-box">DETECTOR</div>
            <p id="subtitle">Instant Skin Health<br></br> Checkup: Upload & Get<br></br> Results</p>
            <p id="note text-danger">Note: This result is AI-generated and not a <br></br>substitute for professional medical advice.<br></br> For further details and personalized treatment<br></br> options, consult your healthcare provider.</p>
          </div>
        </div>

        {/* Right Section */}
        <div id="col-lg-6 right-section">
          <div id="predictions">
            <h2>Prediction: {prediction}</h2>
            <p id="likelihood text-success">Likelihood: {likelihood !== null ? Math.round(likelihood * 100) : 0}%</p>
          </div>

          <div id="upload-section">
              <form onSubmit={handleSubmit} id="upload-form">
                    <div>
                      <input
                        type="file"
                        onChange={handleFileChange}
                        accept=".png,.jpg,.jpeg"
                        required
                        id="file-upload"
                        style={{ display: 'none' }} // Hide the input
                      />
                      <button
                        type="button"
                        onClick={() => document.getElementById('file-upload').click()}
                        id="btn btn-secondary upload-btn"
                        >
                        Upload Image
                      </button>
                      <button
                  type="submit"
                  id="btn bg-CoconutBrown predict-btn"
                >
                  Predict
                </button>
                    </div>
                </form>
          </div>

          {imageUrl && <img src={imageUrl} alt="Uploaded skin" id="uploaded-image" />}
          {error && (
            <div id="error-message">
              <h2>Error: {error}</h2>
            </div>
          )}
```

```
  top: 18%;
  /* font-size: ; */
}

.predict-btn {
  background-color: #4f2f22; /* Custom color */
  border-color: #4f2f22;
  color: #fff;
  font-size: 1.2rem;
  padding: 10px 30px;
  border-radius: 50px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  position: absolute;
  bottom: 2%;
  right: 15%;
}

.predict-btn:hover {
  background-color: #3e261a;
  border-color: #3e261a;
}

.upload-btn {
  background-color: #98613C; /* Custom color */
  border-color: #98613C;
  color: #fff;
  font-size: 1.2rem;
  padding: 10px 30px;
  border-radius: 50px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  margin-right: 10px; /* Adjust spacing if needed */
  position: absolute;
  bottom: 0.9%;
  right: 22%;
}

.upload-btn:hover {
  background-color: #855133;
  border-color: #855133;
}

#custom-heading{
  position: absolute;
  top: 30%;
  font-size: 100px;
  color: #98613C;
  left: 5%;
  top:10%;
  font-family: HelveticaNeueBold;
}

.logo{
  position: absolute;
  top: 0%;
}

.detector-box{
  font-size: 50px;
  font-family: HelveticaNeueRoman;
  color: #471E1A;
  margin: 6%;
  padding: 0px 160px 0px 0px;
  background-color: #E8B693;
  position: absolute;
  top: 30%;
  left: 0%;
}

.note{
  position: absolute;
  top: 75%;
  left: 6%;
  font-size: 20px;
  font-family: HelveticaNeueThin;

}

.subtitle{
  position: absolute;
  top: 55%;
  left: 6%;
  font-family: HansonBold;
  font-size: 20px;
  color: #471E1A;
}
```

```python
import os
import base64
import happybase

# Replace 'localhost' with your HBase server address if it's different
connection = happybase.Connection('localhost', port=9090)

# Define folder names and corresponding table names (dis0 to dis10)
folders_and_tables = {
    "AtopicDermatitis": "dis0",
    "Basal Cell Carcinoma": "dis1",
    "Benign_Keratosis-like_Lesions": "dis2",
    "Eczema": "dis3",
    "Melanocytic_Nevi": "dis4",
    "Melanoma": "dis5",
    "Psoriasis_pictures_Lichen_Planus_and_related_diseases": "dis6",
    "Seborrheic_Keratoses_and_other_Benign_Tumors": "dis7",
    "Tinea_Ringworm_Candidiasis_and_other_Fungal_Infections": "dis8",
    "Warts_Molluscum_and_other_Viral_Infections": "dis9"
}

# Column families
column_families = {
    'image_data': dict()
}

# Define the base directory containing the class folders
base_directory = r"D:\temp\hadtmp"

for folder_name, table_name in folders_and_tables.items():
    # Check if the table already exists, otherwise create it
    if table_name.encode() in connection.tables():
        print(f"Table {table_name} already exists.")
    else:
        # Create the table with column families
        connection.create_table(table_name, column_families)
        print(f"Table {table_name} created successfully with 'image_data' column family.")

    # Define the directory containing the images for the current class
    image_directory = os.path.join(base_directory, folder_name)

    # Access the table
    table = connection.table(table_name)

    # Iterate through the images in the folder
    for image_name in os.listdir(image_directory):
        image_path = os.path.join(image_directory, image_name)
        # Check if it's an image file
        if os.path.isfile(image_path) and image_name.lower().endswith(('png', 'jpg', 'jpeg', 'bmp')):
            # Read the image and encode it in base64
            with open(image_path, 'rb') as image_file:
                encoded_image = base64.b64encode(image_file.read()).decode('utf-8')
            # Insert the encoded image into HBase
            row_key = image_name.split('.')[0]  # Use the image name (without extension) as the row key
            table.put(row_key, {'image_data:image': encoded_image})
            print(f"Inserted image '{image_name}' into HBase table '{table_name}'.")

# Close the connection
connection.close()
```

```python
app = Flask(__name__)
CORS(app)

UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

cnn = load_model(r'D:\swastik\docker\dataorch\cleansed_20_epoch.keras')

class_names = ['Basal Cell Carcinoma',
               'Benign_Keratosis-like_Lesions',
               'Melanocytic_Nevi', 'Melanoma',
               'Seborrheic_Keratoses_and_other_Benign_Tumors',
               'Tinea_Ringworm_Candidiasis_and_other_Fungal_Infections'
               ]


ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

# Kafka producer setup
producer = KafkaProducer(
    bootstrap_servers='localhost:29092',
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def prepare_image(image):
    img = image.convert('RGB')
    img = img.resize((128, 128))
    input_arr = np.array(img)
    input_arr = np.expand_dims(input_arr, axis=0)
    return input_arr

@app.route('/')
def home():
    return '''
        <html>
            <head>
                <title>Skin Disease Prediction API</title>
            </head>
            <body>
                <h1>Skin Disease Prediction</h1>
                <p>Upload an image to get a prediction of possible skin diseases.</p>
            </body>
        </html>
    '''
```

```python
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

class HBaseConnectionError(Exception):
    """Custom exception for HBase connection issues."""
    pass

class DiseasePredictionConsumer:
    def __init__(self, bootstrap_servers=['localhost:29092'], hbase_host='localhost', hbase_port=9090, max_retries=3, retry_delay=5):
        self.bootstrap_servers = bootstrap_servers
        self.hbase_host = hbase_host
        self.hbase_port = hbase_port
        self.max_retries = max_retries
        self.retry_delay = retry_delay
        self.consumer = None
        self.connection = None

        # Initialize connections
        self._initialize_connections()

    def _initialize_connections(self):
        """Initialize Kafka and HBase connections with retry logic."""
        self._init_kafka_consumer()
        self._init_hbase_connection()

    def _init_kafka_consumer(self):
        """Initialize Kafka consumer with retry logic."""
        retry_count = 0
        while retry_count < self.max_retries:
            try:
                self.consumer = KafkaConsumer(
                    'newst',  # New topic for disease predictions
                    bootstrap_servers=self.bootstrap_servers,
                    value_deserializer=lambda x: json.loads(x.decode('utf-8')),
                    auto_offset_reset='earliest',
                    group_id='disease_prediction_group'
                )
                logger.info("Successfully connected to Kafka")
                break
            except Exception as e:
                retry_count += 1
                if retry_count == self.max_retries:
                    raise Exception(f"Failed to connect to Kafka after {self.max_retries} attempts: {str(e)}")
                logger.warning(f"Kafka connection attempt {retry_count} failed. Retrying in {self.retry_delay} seconds...")
                time.sleep(self.retry_delay)

    def _init_hbase_connection(self):
        """Initialize HBase connection with retry logic."""
        retry_count = 0
        while retry_count < self.max_retries:
            try:
                self.connection = happybase.Connection(
                    host=self.hbase_host,
                    port=self.hbase_port,
                    timeout=20000  # Extended timeout
                )
                self.connection.open()  # Explicitly open the connection
                logger.info("Successfully connected to HBase")
                break
            except (socket.error, happybase.hbase.ttypes.IOError) as e:
                retry_count += 1
                if retry_count == self.max_retries:
                    raise HBaseConnectionError(f"Failed to connect to HBase after {self.max_retries} attempts: {str(e)}")
                logger.warning(f"HBase connection attempt {retry_count} failed. Retrying in {self.retry_delay} seconds...")
                time.sleep(self.retry_delay)
```

```python
# Build CNN Model
cnn = tf.keras.models.Sequential()

# Input Layer
cnn.add(tf.keras.layers.InputLayer(input_shape=[128,128,3]))

# First Conv Block
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

# Second Conv Block
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

# Third Conv Block
cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

# Fourth Conv Block
cnn.add(tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=256,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

# Fifth Conv Block
cnn.add(tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=512,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

# Dropout and Dense Layers
cnn.add(tf.keras.layers.Dropout(0.25))
cnn.add(tf.keras.layers.Flatten())
cnn.add(tf.keras.layers.Dense(units=1500,activation='relu'))
cnn.add(tf.keras.layers.Dropout(0.4))

# Output Layer
cnn.add(tf.keras.layers.Dense(units=6,activation='softmax'))  # Assuming 10 classes

# Compile Model
cnn.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
            loss='categorical_crossentropy',
            metrics=['accuracy'])
```