

**A USER-FRIENDLY PLATFORM FOR SKIN DISEASE
IDENTIFICATION AND SELF-EVALUATION WITH DEEP
LEARNING**

**A PROJECT REPORT SUBMITTED TO
SRM INSTITUTE OF SCIENCE & TECHNOLOGY**

**IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR
THE AWARD OF THE DEGREE OF
MASTER OF SCIENCE IN APPLIED DATA SCIENCE**

**BY
ARUNKUMAR A**

REG NO.: RA2232014010056

**UNDER THE GUIDENCE OF
Dr. A. THILAKA M.Sc., M.Phil., Ph.D.,**



**DEPARTMENT OF COMPUTER APPLICATION
FACULTY OF SCIENCE AND HUMANITIES
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR - 603 203
CHENGALPATTU, TAMILNADU**

APRIL 2024

BONAFIDE CERTIFICATE

This is to certify that the project report titled “*A User-friendly Platform for Skin Disease Identification and Self-evaluation with Deep Learning*” is a bonafide work carried by **ARUNKUMAR A (RA2232014010056)** under my supervision for the Degree of Master of Science in Applied Data Science. To my knowledge, the work reported here in is the original work done by the student.

Dr. A. THILAKA

Assistant Professor

Department of Computer Applications

(GUIDE)

Dr. S. ALBERT ANTONY RAJ

Deputy Dean & Head

Department of Computer Applications

Internal Examiner

External Examiner

Declaration of Association of Research Project with SDG Goals

This is to certify that the research project entitled **A USER-FRIENDLY PLATFORM FOR SKIN DISEASE IDENTIFICATION AND SELF-EVALUATION WITH DEEP LEARNING** carried out by **Mr. ARUNKUMAR A** under the supervision Of **Dr. A. THILAKA M.Sc., M.Phil., Ph.D.**, of Department of Computer Applications, in partial fulfilment of the requirement for the award Post Graduation program has been significantly or potentially associated with SDG Goal No **03** titled **GOOD HEALTH AND WELL-BEING**. This study has clearly shown the extent to which its goals and objectives have been met in terms of filling the research gaps, identifying needs, resolving problems, and developing innovative solutions locally for achieving the above-mentioned SDG on a National and/or on an international level.

Signature of the Student

Guide and Supervisor

Head of the Department

ACKNOWLEDGEMENT

With the profound gratitude to the ALMIGHTY, I take this chance to thank people who helped us to complete this project.

I take this as a right opportunity to say THANKS to our Parents who are there to stand with us always with the words “YOU CAN”.

I am thankful to **Dr. T. R. Paarivendhar** Chancellor, SRM Institute of Science & Technology who gave us the platform to establish myself to reach greater heights.

I am also deeply grateful to **Prof. A. Vinay Kumar**, Pro Vice Chancellor of SRM Institute of Science & Technology, whose unwavering support has been instrumental in my journey toward excellence.

I earnestly thank **Dr. A. Duraisamy, Ph.D.**, Dean, College of Science and Humanities, SRM Institute of Science & Technology who always encourage us to do novel things.

I express my sincere thanks to **Dr. S. Albert Antony Raj, Ph.D.**, Deputy Dean and Head, Department of Computer Applications for his valuable guidance and support to execute all incline in learning.

It is our delight to thank our project guide, **Dr. A. THILAKA M.Sc., M.Phil., Ph.D.**, Assistant Professor, Department of Computer Applications for her help, support, encouragement, suggestions and guidance throughout development phases of the project.

I convey our gratitude to all the family members of the department who extended their support through valuable comments and suggestions during the reviews.

A great note of gratitude to friends and people who are known and unknown to us who helped in carrying out this project work a successful one.

ARUNKUMAR A

INTERNSHIP COMPLETION CERTIFICATE



42/3, Kannagi Street, Choolaimedu,
Chennai, Tamil Nadu -600094

Email - support@resnetsolutions.com | Contact Number - +91-994-487-8589

CIN: U62099TN2023PTC164150

TO WHOMSOEVER IT MAY CONCERN

This certificate is hereby awarded to **Mr. ARUNKUMAR A**, with Registration Number RA2232014010056, a diligent student pursuing Master of Science in **Applied Data Science** at **SRM Institute of Science and Technology, Kattangulattur**, has successfully completed as an **"ML & DL Intern"** at **ResNetSolution Pvt. Ltd.**

This certificate serves as confirmation of **Mr. ARUNKUMAR A**'s **successful completion** of the internship and project titled **"A User-friendly Platform for Skin Disease Identification and Self-evaluation with Deep Learning"** at our esteemed organization. The internship spanned from 05/02/2024 to 15/03/2024, throughout his internship, **Mr. ARUNKUMAR A** actively engaged in various tasks and processes integral to the project, **showcasing his punctuality, hard work, and inquisitive nature**. His proactive approach and commitment to excellence have significantly contributed to the success of the project.

In **recognition of his outstanding performance and dedication**, we are pleased to present this certificate to **Mr. ARUNKUMAR A**. We believe that his experiences gained during this internship will serve as a solid foundation for his **future endeavours in the field of ML & DL**.

Congratulations to **Mr. ARUNKUMAR A** on this remarkable achievement. We wish him continued success in all his future academic and professional pursuits.

B. Kesaven B

Kesaven B

Founder & CEO



Thanks & Regards

Resnet Solution Private Limited

Chennai-600 094

PLAGIARISM CERTIFICATE

A User-friendly Platform for Skin Disease Identification and Self-evaluation with Deep Learning

ORIGINALITY REPORT

0%

SIMILARITY INDEX

0%

INTERNET SOURCES

0%

PUBLICATIONS

%

STUDENT PAPERS

PRIMARY SOURCES

Exclude quotes Off

Exclude bibliography Off

Exclude matches < 10 words

Submission date: 05-Apr-2024 12:05PM (UTC+0530)

Submission ID: 2340545197

File name: Arun.pdf (127.38K)

Word count: 184

Character count: 1164

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
I	INTRODUCTION 1.1 OVERVIEW OF PROJECT 1.2 SCOPE OF PROJECT 1.3 OBJECTIVE OF PROJECT 1.4 IMPORTANCE OF EARLY IDENTIFICTION 1.5 CURRENT CHALLENGES IN DIAGNOSIS 1.6 ROLE OF DEEP LEARNING IN IMPROVING DETECTION	1
II	LITERATURE SURVEY	6
III	SOFTWARE REQUIREMENT ANALYSIS 3.1 HARDWARE REQUIREMENTS 3.2 SOFTWARE REQUIREMENTS 3.3 LANGUAGE SPECIFICATION – PYTHON 3.4LIBRARIES USED	11
IV	SYSTEM STUDY 4.1 EXISTING SYSTEM 4.1.1 Disadvantages of Existing System 4.2 PROPOSED SYSTEM 4.2.1 Advantages of Proposed System 4.3 FEASIBILITY STUDY	21
V	SYSTEM DESIGN 5.1 SYSTEM ARCHITECTURE 5.2 USE CASE DIAGRAM 5.3 ACTIVITY DIAGRAM 5.4 SEQUENCE DIAGRAM 5.5 CLASS DIAGRAM	25

	5.6 ER DIAGRAM 5.7 DATA FLOW DIAGRAM	
VI	MODULE DESCRIPTION 6.1 IMAGE SOURCES 6.2 DATA ANNOTATION 6..3 ARCHITECTURE OF CNN 6.4 SCALABILITY AND MAINTENANCE 6.5. IMAGE PREPROCESSING TECHNIQUES 6.6 DATA AUGMENTATION AND IMAGE ENHANCEMENT 6.6.1 Data Augmentation 6.6.2 Image Enhancement 6.7 FEATURE EXTRACTION AND SELECTION 6.7.1 Feature Extraction 6.7.2 Feature Selection 6.8. DATA NORMALIZATION AND SPLITTING 6.8.1 Data Normalization 6.8.2 Data Splitting	34
VII	MODEL IMPLEMENTATION 7.1 REQUIREMENT ANALYSIS AND SPECIFICATION 7.1.1 Input Data 7.1.2 Input Data 7.2 PROJECT REQUIREMENTS 7.2.1 Functional Requirements 7.2.2 Non-Functional Requirements 7.3 PROJECT WORKFLOW 7.4 STEPS INVOLVED IN SKIN DISEASES IDENTIFICATION 7.4.1 Data Collection 7.4.2 Preparing Data	50

	7.4.3 Making Predictions 7.4.4 Building Model 7.4.4.1 Splitting Dataset 7.4.4.2 Convolutional Neural Network 7.5 MODEL SELECTION 7.6 TRAINING PROCESS AND TUNING PARAMETERS 7.6.1 Training Process 7.6.2 Parameter Tuning 7.7 EVALUATION METRIC AND PERFORMANCE ANALYSIS	
VIII	TESTING 8.1 TYPES OF TESTING 8.1.1 Unit Testing 8.1.2 Integration Testing 8.1.3 Functional Testing	64
IX	MODEL DEPLOYMENT 9.1 IMPORT THE GIVEN IMAGE FROM DATASET 9.2 DEPLOYING THE MODEL IN STREAMLIT FRAMEWORK AND HTML 9.2.1 Streamlit 9.2.2 Html 9.3 DATABASE 9.4 SECURITY	69
X	CONCLUSION 10.1 RESULT 10.2 FURTHER ENHANCEMENTS	78
	BIBLIOGRAPHY	81
	APPENDIX – I SCREENSHOT	83
	APPENDIX – II SOURCE CODE	88

ABSTRACT

This project presents the development of a skin disease identification system using deep learning techniques, particularly convolutional neural networks (CNNs), to automate the detection and classification of various skin conditions based on image data. By training the CNN model on a diverse dataset of skin images, the system learns to recognize patterns and features indicative of different skin diseases, including melanoma, eczema, and psoriasis. Through feature extraction and classification, the model can accurately diagnose skin diseases. The proposed system offers a non-invasive and cost-effective approach to skin disease diagnosis, facilitating early intervention and timely treatment. Additionally, a user-friendly interface is created to host the system, allowing users to create accounts, upload skin images, and receive reports containing the identified disease along with preventative and precautionary measures. This implementation of deep learning for skin disease identification has the potential to enhance healthcare outcomes by enabling faster and more accurate diagnosis, ultimately improving patient care and quality of life.

LIST OF ACRONYMS AND ABBREVIATIONS

Sno.	Acronyms	Abbreviations
1)	ML	Machine Learning
2)	DL	Deep Learning
3)	CNN	Convolutional Neural Network
4)	ANN	Artificial Neural Network
5)	GUI	Graphical User Interface
6)	API	Application Programming Interface
7)	IOT	Internet of Things
8)	UI	User Interface
9)	GPU	Graphics Processing Unit
10)	CPU	Central Processing Unit
11)	OS	Operating System

LIST OF FIGURES

S No.	Figure No.	Figure Name	Page No.
1	5.1	System Architecture	26
2	5.2	Use Case Diagram	27
3	5.3	Activity Diagram	28
4	5.4	Sequence Diagram	29
5	5.5	Class Diagram	30
6	5.6	Er Diagram	31
7	5.7	Data Flow Diagram(L1)	32
8	5.8	Data Flow Diagram(L2)	33
9	6.1	Architecture Of CNN	36
10	6.2	Pre-Preprocessing	40
11	6.3	Feature Extraction	44
12	6.4	Data Splitting	48
13	7.1	Flow of Pre-processing Training the CNN Model	54
14	7.2	Flowchart for the Emotion Identification	54
15	7.3	Construction of Detecting Mode	55

CHAPTER I

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF PROJECT

Skin disease Identification with the help of deep learning is an emergent field that depends on advanced algorithms and computational methods to detect and diagnose diverse skin disorders. This new methodology relies on examining pictures of the skin, for example dermatoscopic images or photos, in order to make out patterns or features which can be linked to various types of skin diseases.

Large labelled datasets containing dermatitis, melanoma, psoriasis among others is used when training deep learning models so that they can automatically differentiate between healthy skins and different skins conditions. By extracting features from these images and using classification algorithms, such models can achieve high accuracy in diagnosing skin diseases.

The combination of machine learning and deep learning techniques in identifying skin diseases not only ensures quick diagnoses but it also has potential for supporting healthcare providers in making better treatment decisions. Moreover, use of deep learning algorithms can also aid traditional limitations in diagnosis of skin diseases like subjective human judgment variability as well as need for specialized expertise.

This project aims to detect these seven classes of skin cancer utilizing Convolutional Neural Network (CNN) architectures implemented with the Keras TensorFlow backend. Subsequently, the results will be analyzed to evaluate the practical utility of the model in real-world scenarios. The technical validation component involves the examination of the distribution of ground truth data, specifically the "dx_type" field, which delineates four categories: Histopathology (Histo), Confocal, Follow-up, and Consensus.

The model architecture is constructed using the Keras Sequential API, comprising dense layers, dropout regularization, rectifier activation functions, and flatten layers. The optimization process involves the selection of the Adam optimizer, renowned for its efficacy in deep learning tasks due to its adaptive gradient and root mean square propagation characteristics. The model's performance is evaluated using the accuracy metric to gauge its effectiveness in classifying skin lesions across multiple categories.

1.2 SCOPE OF PROJECT

The project for skin disease identification using deep learning aims to address a critical need in healthcare by developing a robust system capable of accurately identifying various types of skin diseases. At its core, this system will leverage advanced deep learning algorithms to analyze the features present in skin lesions, providing a reliable classification or diagnosis of the condition. By harnessing the power of deep learning, the system will be able to process and interpret complex image data with a high degree of accuracy, aiding healthcare professionals in making informed decisions regarding patient care.

Central to the success of this project is the collection and labelling of a diverse dataset of skin disease images. This dataset will serve as the foundation for training and testing the deep learning model, ensuring its ability to accurately recognize and classify different skin conditions. The dataset will encompass a wide range of skin diseases, including common dermatological issues as well as rarer conditions, to ensure the model's versatility and effectiveness across various scenarios.

Ultimately, the project's overarching goal is to develop a user-friendly application that can be easily accessed and utilized by healthcare professionals. This application will serve as a valuable tool in clinical settings, enabling practitioners to swiftly and accurately diagnose skin diseases based on uploaded images. By providing timely and accurate diagnoses, the system will empower healthcare professionals to deliver prompt treatment recommendations, ultimately improving patient outcomes and enhancing overall healthcare delivery in the realm of dermatology.

1.3 OBJECTIVE OF PROJECT

The project aims to revolutionize the diagnosis of skin diseases through the development of a sophisticated deep learning model. Leveraging image processing techniques and cutting-edge deep learning algorithms, the system will accurately analyse and classify dermatological images into distinct categories of skin diseases. By providing healthcare professionals with a powerful tool for automated analysis, the project seeks to enhance the efficiency and accuracy of skin disease diagnosis. This technology holds the

promise of significantly improving early identification and treatment of various skin conditions, thereby mitigating potential health risks and improving patient outcomes.

Through meticulous research and development, the project endeavors to bridge the gap between traditional diagnostic methods and advanced technological solutions. By harnessing the capabilities of deep learning, the model aspires to not only streamline the diagnostic process but also empower healthcare professionals with valuable insights into skin diseases. With its potential to offer rapid and precise analysis, the envisioned tool has the capacity to revolutionize dermatological healthcare practices, ensuring timely interventions and personalized treatment plans. Ultimately, the project's overarching goal is to democratize access to accurate skin disease diagnosis, ushering in a new era of proactive healthcare management.

1.4IMPORTANCE OF EARLY IDENTIFICTION

Early detection is important for the prevention and treatment of skin diseases as it can improve outcomes and reduce healthcare costs. Using deep learning to quickly diagnose skin diseases and accurately pinpoint potential problems, enabling timely intervention and management Through large databases and systems on the skin with image analysis, deep learning systems can detect changes that are not easily detected by the human eye

Early detection through in-depth learning can also help monitor disease progression and treatment effectiveness, leading to personalized care planning and improved outcomes for patients Furthermore, the ability to diagnose skin diseases early can help reduce the burden on health care systems by preventing potentially lengthy and costly interventions. Overall, the use of deep learning for early detection of skin diseases is beneficial not only for individual patients but also for public health and health resource allocation.

1.5CURRENT CHALLENGES IN DIAGNOSIS

One of the current challenges in implementing deep learning for skin diagnosis is the lack of diverse and well-documented data. In addition to the limited availability of high-quality data encompassing multiple skin types, age groups, and conditions, which

prevents the development of accurate generalizable models, changes and trends in dermatology pose another major challenge. Skin conditions can manifest in a variety of ways and can visually distinguish even experienced dermatologists. Thus, the ability of deep learning systems to effectively capture and interpret this complex visual data remains a significant obstacle.

Finally, for skin diagnosis, the ability to interpret deep learning models and the interpreter is another important challenge. Ensuring that the decision-making process for these models is transparent and understood by health professionals is essential to their adoption in clinical practice. Addressing these challenges will be key to promoting the use of machine learning for accurate and reliable diagnosis of skin diseases.

1.6ROLE OF DEEP LEARNING IN IMPROVING DETECTION

Deep learning plays a key role in improving dermatological diagnosis through systems and techniques that use big data to improve accuracy and performance. By providing training programs on a variety of advanced levels, deep learning algorithms can identify patterns and trends in skin images. In this regard, the main advantage of deep learning is the classification of skin lesions based on visual characteristics such as colour, texture, and shape, which can be difficult for human observers in presence.

Furthermore, through extensive learning and adaptive adaptation, deep learning models can evolve and improve their performance over time, increasing accuracy and reliability throughout the rise in dermatology. Overall, the integration of deep learning into dermatology has great potential to revolutionize diagnostic strategies.

CHAPTER II

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

[1] Nawal Soliman ALKolifi ALEnezi, Procedia Computer Science 163 (2019) 85–92A Method of Skin Disease Detection Using Image Processing and Machine Learning.

The paper titled "A Method of Skin Disease Detection Using Image Processing and Machine Learning" by Nawal Soliman ALKolifi ALEnezi, presented at the 16th International Learning & Technology Conference 2019, addresses the pressing need for efficient and accessible methods of diagnosing skin diseases. Despite advancements in medical technology, the affordability and accessibility of skin disease diagnostics remain limited. Leveraging image processing techniques, the study proposes a cost-effective and accurate approach to automate the detection process. By utilizing readily available tools like digital cameras and standard computing equipment, the method involves acquiring digital images of affected skin areas, preprocessing them to enhance clarity, extracting features using a pretrained convolutional neural network, and classifying these features using a Multiclass Support Vector Machine. Results demonstrate a remarkable accuracy rate of 100% in detecting various types of skin diseases, providing valuable information on disease type, spread, and severity to users. This innovative approach not only ensures reliable diagnosis but also minimizes the reliance on specialized equipment and expertise, holding promise for improving dermatological healthcare delivery, particularly in resource-constrained settings.

[2] An automated detection of Scabies skin disease Using Image Processing and CNN - Monishanker Halder*, Hussain Moh. Emrul Kabir, Afsana Mimi Rity, and Arobindo Vowmik.DOI: 10.1109/STI56238.2022.10103250

In every year, human faces several types of skin diseases. Bacteria, fungal infections, viruses, allergies, etc. are the reason for different types of skin diseases. Many people don't show interest in identifying or treatment for skin diseases because of the cost of diagnosis which is expensive and lack of concentration. Image processing is one the easiest way to detect skin diseases by classifying the image of the affected area. A dermatologist can easily identify the problem by using the image processing technique.

Feature extraction helps to classify skin diseases effectively. In this paper, we focused on Scabies which is one of the most common skin diseases. We proposed a methodology using image processing and CNN to detect scabies. Here thresholding is used for segmenting the affected area and CNN is used for classifying images. We used different types of data augmentation techniques for increasing data where each of the images was considered unique. Our proposed method is simple and easy to use. In this research, RGB images are used and we made a dataset where all images are collected from different sources in the internet for training the system. The proposed methodology can detect scabies with an accuracy of 97.25%.

[3] Analysis on Convolutional Neural Network Model using Skin Disease Dataset, Proceedings of the International Conference on Sustainable Computing & Smart Systems (ICSCSS 2023) IEEE Xplore Part No: CFP23DJ3-ART; ISBN: 979-8-3503-3360-2

Skin conditions are most frequently seen in persons of all ages and are brought on by bacteria, infections, or radiation. Some illnesses spread over time and have several harmful skin effects. Extreme weather, pollution, and depletion of the ozone layer are a few effects of climate change that could damage the skin condition. Skin diseases can be cured in over 90% of cases if they are discovered and treated in the early stages. The convolutional neural network approach is an effective method for identifying skin disorders that may lead to better outcomes for both treatment and diagnosis. In this study, the author uses the convolutional neural network (CNN) technique because of its great precision in extracting characteristics from pictures and their ability to recognize patterns, CNNs are used for picture analysis and identification and provide a review of deep learning algorithm and how they can be used to diagnose skin conditions. simple description of skin conditions, image acquisition and a literature review on deep learning techniques in dermatology presented by different authors. When compared to VGG16, Resnetv2 and the CNN model, the CNN model is better because its computation effectiveness, capacity to identify spatial layouts, flexibility, and interpretation. The visual signs of several skin conditions can appear identical, making it difficult for CNN models to differentiate among diseases. CNN-based models may not apply adequately to different datasets after being trained on one dataset. The results show that the suggested technique has a higher level of diagnostic precision. It improves the effectiveness of an CNN model

that has been pre-trained for detecting certain types of skin illnesses by using transfer learning along with it.

[4] Shuchi Bhadula, Sachin Sharma, Piyush Juyal, Chitransh Kulshrestha Machine Learning Algorithms based Skin Disease Detection, International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075 (Online), Volume-9 Issue-2, December 2019

Skin disease recognition and observing is a major challenge looked by the medical industry. Because of expanding contamination and utilization of lousy nourishment, the tally of patients experiencing skin related issues is expanding at a quicker rate. Well-being isn't the main concern, however unfortunate skin hurts our certainty. Customary and appropriate skin checking is a significant advance towards early discovery of any destructive or starting changes in skin that may bring about skin disease. Machine learning methods can add to the improvement of capable frameworks which can order various classes of skin illnesses. To identify skin maladies, first, it is required to separate the skin and non-skin. In this paper, five diverse machine learning algorithms have been chosen and executed on skin infection data set to anticipate the exact class of skin disease. Out of a few machine learning algorithms, we have worked on Random forest, naive bayes, logistic regression, kernel SVM and CNN. A similar examination dependent on confusion matrix parameters and training accuracy has been performed and delineated utilizing graphs. It is discovered that CNN is giving best training precision for the right expectation of skin diseases among all selected.

[5] Classification and detection of skin diseases basedon CNN-powered image segmentation, 2023 3rd International Conference on Intelligent Technologies (CONIT)Karnataka, India. June 23-25, 2023

Skin diseases have become more prevalent in recent years, posing a significant burden on healthcare system globally. These conditions can range from noncancerous problems like acne, eczema, and vitiligo to more severe diseases such as melanoma, basal and squamous cell carcinoma. Skin ailments not only have an impact on physical health but can also affect the patient's psychological well-being, especially when the face is disfigured or damaged. Early detection and diagnosis of dermatological conditions can significantly improve survival rates as their severity can worsen over time. This article

covers various methods used to identify and diagnose skin diseases, including natural diagnostic techniques and image pre-processing mechanisms and classification algorithms used globally. By understanding these techniques, healthcare professionals can detect skin diseases accurately, leading to better patient outcomes. Advancements in technology and research in this field continue to progress, providing hope to those who suffer from skin diseases.

[6] Skin Disease Detection and Recommendation System using Deep Learning and Cloud Computing. Proceedings of the 8th International Conference on Communication and Electronics Systems (ICCES 2023) IEEE Xplore Part Number: CFP23AWO-ART; ISBN: 979-8-3503-9663-8

The main objective of this research is to develop an application based on Deep learning, Computer vision and cloud computing that detects the different kinds of skin diseases caused by different types of viruses, Bacteria, Fungus and Environment. This study has also developed and integrated a recommendation system, which recommends the medicines and care taking process for a particular disease. The application also suggests preventive methods for different kinds of skin infections. This study used an ensemble of convolution neural networks (CNN) with generative adversarial network (GAN) and Computer vision for construction of the model. Further, Amazon Personalize is used to build recommendation system in the proposed web application. The proposed application detects the disease based on symptoms, pictures, and videos of infected skin area. The application will be helpful for dermatologists and common people to perform early detection and prevention of skin diseases in India. This study also compared the accuracy of ensemble of convolution neural networks (CNN) with GAN and other algorithms like CNN. In comparison of accuracy, this study found that the Ensembles of CNN with GAN give best results for the proposed dataset.

CHAPTER III

SOFTWARE REQUIREMENT

ANALYSIS

CHAPTER 3

SOFTWARE REQUIREMENT ANALYSIS

3.1 HARDWARE REQUIREMENTS

Processor	:	Intel core2/Ryzen3 or More
Memory	:	4 GB RAM or More
Hard disk Requirement	:	Free 500GB on installation drive

3.2 SOFTWARE REQUIREMENTS

- **Operating system:** Windows7 (with service pack 1) and above OS
- Browser with latest version Installed with Good Web Application Support
- **Language:** Python – Machine Learning and Deep Learning
- **Platform:** Google Colab and Visual Studio Code
- **Framework :** TensorFlow – Keras

3.3 LANGUAGE SPECIFICATION – PYTHON

Among programmers, Python is a favourite because of its ease of use, beautiful features, and versatility. Python is a very suitable programming language for machine learning and deep learning because it can run on its own platform and is widely used by the user community. Machine learning is a branch of AI that aims to eliminate the need for explicit programming by allowing computers to learn from their mistakes and simply perform routine tasks but "artificial intelligence" (AI) includes a broader definition of "machine learning," and eventually train them to make important decisions for themselves.

Deep learning is a microscopic form of artificial intelligence that mimics human brain functions that process data and develop algorithms to be used in decision-making. It uses an artificial neural network consisting of multiple connected neurons, and enables the extraction of elements from raw data to be automatically used. It has happened. Its ability to process large amounts of unstructured data has led to significant advances in image recognition, speech translation, medical diagnosis, etc. With continued research and development, deep learning goes so far as to push the boundaries of what's possible in machine learning and AI applications.

The desire to find intelligent solutions to real problems has further led to the need for AI to automate complex tasks without AI. These advances are essential to solving real problems in the search for intelligent solutions. Python is a widely used programming language that is generally considered to be the best algorithm to help automate such things. Python offers great simplicity and robustness compared to other programming languages. Additionally, the presence of a dynamic Python community makes it easy for programmers to discuss ongoing projects and make recommendations on how to make their programs more efficient.

ADVANTAGES OF USING PYTHON

- Variety of Framework and libraries:

A good programming environment requires libraries and frameworks. Python frameworks and libraries simplify programme development. Developers can speed up complex project coding with prewritten code from a library. PyBrain, a modular machine learning toolkit in Python, provides easy-to-use algorithms. Python frameworks and libraries provide a structured and tested environment for the best coding solutions.

- Reliability

Most software developers seek simplicity and consistency in Python. Python code is concise and readable, simplifying presentation. Compared to other programming languages, developers can write code quickly. Developers can get community feedback to improve their product or app. Python is simpler than other programming languages, therefore beginners may learn it quickly. Experienced developers may focus on innovation and solving real-world problems with machine

learning because they can easily design stable and trustworthy solutions.

- Easily Executable

Developers choose Python because it works on many platforms without change. Python runs unmodified on Windows, Linux, and macOS. Python is supported on all these platforms; therefore, you don't need a Python expert to comprehend it. Python's great executability allows separate applications. Programming the app requires only Python. Developers benefit from this because some programming languages require others to complete the job. Python's portability cuts project execution time and effort.

3.4 LIBRARIES USED

- | | |
|---------------|--------------|
| • NumPy | • Json |
| • Pandas | • Glob |
| • Matplotlib | • PIL |
| • Seaborn | • Shap |
| • Scikitlearn | • Keras |
| • OS | • Tensorflow |
| • IO | • Streamlit |

NUMPY:

- NumPy is a Python library for scientific computing.
- It provides a high-performance implementation of multidimensional arrays and matrices, as well as a large collection of mathematical functions for operating on those arrays.
- NumPy is widely used in data science, machine learning, and scientific computing.

Features:

- *Multidimensional arrays:* NumPy provides a high-performance implementation of multidimensional arrays, which are essential for many scientific and engineering applications.

- *Mathematical functions:* NumPy provides a large collection of mathematical functions for operating on multidimensional arrays, including linear algebra, statistical, and Fourier transform functions.
- Broadcasting: NumPy provides a powerful broadcasting mechanism that allows for efficient operations on arrays of different shapes.

PANDAS:

- Pandas is a Python library for data analysis.
- It is built on top of the NumPy library and provides high-level data structures and operations for manipulating numerical data and time series.
- Pandas is widely used in data science, machine learning, and financial analysis.

Features:

- *Data structures:* Pandas provides two main data structures for storing and manipulating data: Series and DataFrame. A Series is a one-dimensional labeled array, similar to a Python list. A DataFrame is a two-dimensional labeled array, similar to a Python dictionary.
- *Data manipulation:* Pandas provides a wide range of functions for manipulating data, including sorting, filtering, grouping, and aggregating.
- *Data visualization:* Pandas integrates with Matplotlib, a Python library for data visualization, to provide easy-to-use plotting functions.
- *Time series analysis:* Pandas provides a variety of tools for working with time series data, such as date parsing, time shifting, and resampling.

SEABORN:

- Seaborn is a Python data visualization library built on top of Matplotlib.
- It provides a high-level interface for creating attractive and informative statistical graphics.
- Seaborn is widely used in data science, machine learning, and statistical analysis.

Features:

- *Ease of use:* Seaborn has a simple and intuitive interface, making it easy to learn and use.
- *Attractive and informative visualizations:* Seaborn produces high-quality visualizations with a consistent aesthetic.
- *Statistical integration:* Seaborn provides a variety of functions for statistical analysis, such as regression, correlation, and hypothesis testing.
- *Deep integration with Pandas:* Seaborn is tightly integrated with Pandas, making it easy to create visualizations from Pandas Data Frames.

MATPLOTLIB:

- Matplotlib is a Python library for data visualization.
- It provides a wide range of plotting functions for creating static, animated, and interactive visualizations.
- Matplotlib is widely used in data science, machine learning, and scientific computing.

Features:

- *Comprehensive set of plotting functions:* Matplotlib provides a wide range of plotting functions for creating line plots, bar plots, scatter plots, histograms, and many other types of visualizations.
- *Easy to use:* Matplotlib has a simple and intuitive interface, making it easy to learn and use.
- *Flexible and customizable:* Matplotlib is highly customizable, allowing users to create visualizations that meet their specific needs.
- *Well-documented:* Matplotlib is well-documented, with a comprehensive user guide and tutorials.

SCIKITLEARN:

- Scikit-learn is a free software machine learning library for the Python programming language.

- It features various classification, regression, clustering and dimensionality reduction algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- Scikit-learn is widely used in data science, machine learning, and natural language processing.

Features:

- *Comprehensive set of algorithms:* Scikit-learn provides a wide range of machine learning algorithms for classification, regression, clustering, and dimensionality reduction.
- *Easy to use:* Scikit-learn has a consistent and user-friendly interface, making it easy to learn and use.

OS:

- The os module in Python provides a portable way of using operating system-dependent functionality.
- It allows Python programs to interact with the underlying operating system in a platform-independent manner, making it easier to write cross-platform applications.
- The os module provides various functions for file and directory operations, process management, environment variables, and more.

Features:

- *Portability:* While the os module strives for portability, some functions might exhibit slight behavioral differences across operating systems. Consider this when writing code that needs to work seamlessly across various platforms.
- *Caution:* Be mindful when modifying the file system or environment variables. Incorrect operations can lead to unintended consequences or data loss.
- *Abstractions:* For higher-level file and directory management, explore the shutil module. The os.path submodule offers

SHAP:

- SHAP is a powerful Python library designed to enhance the interpretability of machine learning models.
- It accomplishes this by calculating SHAP values, which represent the contribution of each feature in a dataset to a model's prediction.
- This allows you to understand how individual features influence the model's output and gain valuable insights into its decision-making process.

Features:

- *Increased Transparency:* SHAP sheds light on your model's "black box," enabling you to identify potential biases or fairness issues. This enhanced understanding can help build trust in your models.
- *Feature Engineering:* The information gleaned from SHAP can inform feature engineering strategies, where new features are created to capture complex relationships or interactions better.

STREAMLIT:

Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for data science and machine learning. With just a few lines of Python code, you can create interactive data visualizations, dashboards, and machine learning models that can be shared with others.

Features:

- *Ease of use:* Streamlit is designed to be easy to use, even for those with no prior experience in web development. Simply add Streamlit commands to your Python code, and Streamlit will automatically generate a web app based on your code.
- *Interactivity:* Streamlit apps are highly interactive, allowing users to explore data and experiment with machine learning models in real time. This makes Streamlit a great tool for communicating data science and machine learning results to others.

- *Customization:* Streamlit apps can be customized to meet your specific needs. You can add your own branding, layouts, and components to create a truly unique user experience.
- *Sharing:* Streamlit apps can be easily shared with others. Simply deploy your app to Streamlit Cloud or share the code with others so they can run it locally.

TENSORFLOW:

- TensorFlow is a powerful open-source library developed by Google for machine learning and artificial intelligence.
- It's particularly well-suited for tasks involving deep neural networks, which are a type of artificial neural network inspired by the structure and function of the human brain.

Features:

- *Keras Integration:* TensorFlow offers a high-level API called Keras, which simplifies the process of building and training models. Keras provides pre-built building blocks for neural networks and streamlines common machine learning tasks.
- *Flexibility:* TensorFlow can be used for various machine learning applications, not just deep learning. It supports a wide range of algorithms and allows for customization at different levels.
- *Hardware Acceleration:* TensorFlow leverages various hardware platforms, including CPUs, GPUs, and TPUs (Tensor Processing Units), to accelerate training and inference. This allows you to train models faster and deploy them on different devices.
- *Large Ecosystem:* TensorFlow has a vast ecosystem of tools and resources, including:
 - TensorBoard: A visualization tool for monitoring and debugging your machine learning models.
 - TensorFlow Hub: A repository of pre-trained models that you can use as a

starting point for your own projects.

- TensorFlow Lite: A framework for deploying machine learning models on mobile and edge devices.

KERAS:

- Keras is a powerful and popular open-source library written in Python that simplifies building and training deep learning models.
- It acts as a high-level interface for various backend libraries like TensorFlow, Theano, or CNTK, allowing you to focus on the high-level design of your neural networks without getting bogged down in low-level details.

Features:

- *Fast Experimentation:* The high-level abstractions in Keras accelerate the process of prototyping and iterating on deep learning models. You can quickly test different configurations without getting entangled in low-level code.
- *Backend Agnosticism:* Keras doesn't force you to use a specific backend engine. You can switch between TensorFlow, Theano, or CNTK depending on your needs and preferences. This flexibility ensures you can leverage the strengths of different backends for diverse deep learning tasks.
- *Scalability:* Keras models can run on a variety of hardware platforms, including CPUs, GPUs, and TPUs (Tensor Processing Units), allowing you to scale your deep learning projects to handle large datasets and complex models.

CHAPTER IV

SYSTEM STUDY

CHAPTER 4

SYSTEM STUDY

4.1 EXISTING SYSTEM

Skin diseases are common ailments affecting millions of people worldwide, ranging from benign conditions like acne to severe illnesses such as melanoma. Traditionally, diagnosing skin diseases has heavily relied on visual examination by dermatologists, which can be time-consuming and subjective. Moreover, accessibility to dermatologists may be limited in certain regions, leading to delayed diagnosis and treatment. In the existing system, manual inspection and diagnosis by dermatologists are the primary methods used for identifying skin diseases. Dermatologists examine patients' skin visually, looking for characteristic patterns, colours, textures, and other visual cues associated with different skin conditions. This process is subjective and prone to errors, as it relies heavily on the expertise and experience of the dermatologist. Additionally, the availability of dermatologists may vary, leading to inconsistencies in diagnosis and delays in treatment. Furthermore, the existing system lacks scalability and efficiency, especially in areas with a high demand for dermatological services. Patients may face long wait times for appointments, and the process of diagnosis and treatment can be prolonged, impacting their quality of life and potentially worsening their condition. Moreover, misdiagnosis or delayed diagnosis can have serious consequences, including the progression of diseases to more advanced stages or unnecessary treatments.

4.1.1 DISADVANTAGES OF EXISTING SYSTEM

Subjectivity: Diagnosis relies heavily on the subjective interpretation of visual cues by dermatologists, leading to inconsistencies and errors.

Limited Access: Access to dermatologists may be limited, especially in rural or underserved areas, leading to delays in diagnosis and treatment.

Inefficiency: Manual diagnosis is time-consuming and may result in long wait times for patients, impacting their overall healthcare experience.

Error-prone: Human errors in diagnosis may occur due to the complexity and variability of skin diseases, leading to misdiagnosis or delayed diagnosis.

4.2 PROPOSED SYSTEM

To address the limitations of the existing system, a proposed system for skin disease identification using deep learning offers a promising solution. Leveraging advancements in artificial intelligence and computer vision, the proposed system aims to automate and enhance the process of skin disease diagnosis, making it more accurate, efficient, and accessible.

The proposed system utilizes deep learning algorithms trained on large datasets of annotated skin images to recognize patterns and features indicative of various skin diseases. By analysing digital images of patients' skin, the system can identify potential abnormalities and provide preliminary diagnoses or recommendations for further evaluation by healthcare professionals.

4.2.1 ADVANTAGES OF PROPOSED SYSTEM

Accuracy: Machine learning algorithms can analyse skin images objectively and consistently, reducing the risk of human errors and improving diagnostic accuracy.

Accessibility: The proposed system can be deployed in various healthcare settings, including remote or underserved areas, enabling broader access to dermatological services.

Efficiency: Automated analysis of skin images speeds up the diagnosis process, reducing wait times for patients and improving overall healthcare efficiency.

Early Detection: By detecting skin diseases at an early stage, the proposed system can facilitate timely intervention and treatment, potentially improving patient outcomes and reducing healthcare costs.

Scalability: The scalability of machine learning algorithms allows the proposed system to handle large volumes of data and accommodate growing demand for dermatological

4.3 FEASIBILITY STUDY

A feasibility study assessing various aspects to determine the project's viability. Factors to consider include technical feasibility, operational feasibility, economic feasibility, legal and ethical considerations, and schedule feasibility. From a technical perspective, the study will evaluate the availability of data sources, the complexity of implementing deep learning algorithms, and the compatibility with existing systems.

Operational feasibility will involve determining if the system meets the needs of users, is user-friendly, and can be integrated into existing workflows seamlessly. Economic feasibility will assess the costs associated with system development, maintenance, and training, as well as the potential benefits and cost savings that could result from implementing the system. Legal and ethical considerations will focus on ensuring compliance with data privacy regulations, ethical use and transparency in decision-making processes.

Finally, schedule feasibility will involve defining a realistic timeline for system development, testing, and deployment to ensure that it aligns with stakeholder expectations and project deadlines. Overall, the feasibility study for this project will be crucial in identifying potential challenges, risks, and opportunities to ensure the successful development and implementation for developing a Skin Disease Identification System using deep learning, a comprehensive feasibility study would be essential.

Technical aspects such as data availability, algorithm complexity, and system compatibility will be evaluated, while operational considerations will focus on user needs, integration with existing processes, and user-friendliness. Economic feasibility will assess development and maintenance costs, potential cost savings, and benefits. Legal and ethical aspects will ensure compliance with regulations, ethical AI use, and transparent decision-making.

Schedule feasibility will define a timeline for development, testing, and deployment to meet stakeholder expectations and deadlines, ultimately ensuring the successful implementation of a skin disease identification system leveraging deep learning technologies.

CHAPTER V

SYSTEM DESIGN

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

System architecture defines the blueprint of a system, detailing the arrangement and function of its components to achieve operational goals. It includes hardware, software, networks, and interfaces with defined interactions. Effective system architecture is scalable, reliable, secure, and efficient, promoting seamless communication and collaboration. By fostering integration and smooth operation, it optimizes performance and functionality. A robust system architecture underpins system success, ensuring adaptability and coherence for long-term sustainability and effectiveness. In essence, system architecture is the foundation that empowers a system to deliver its intended outcomes efficiently and dependably.

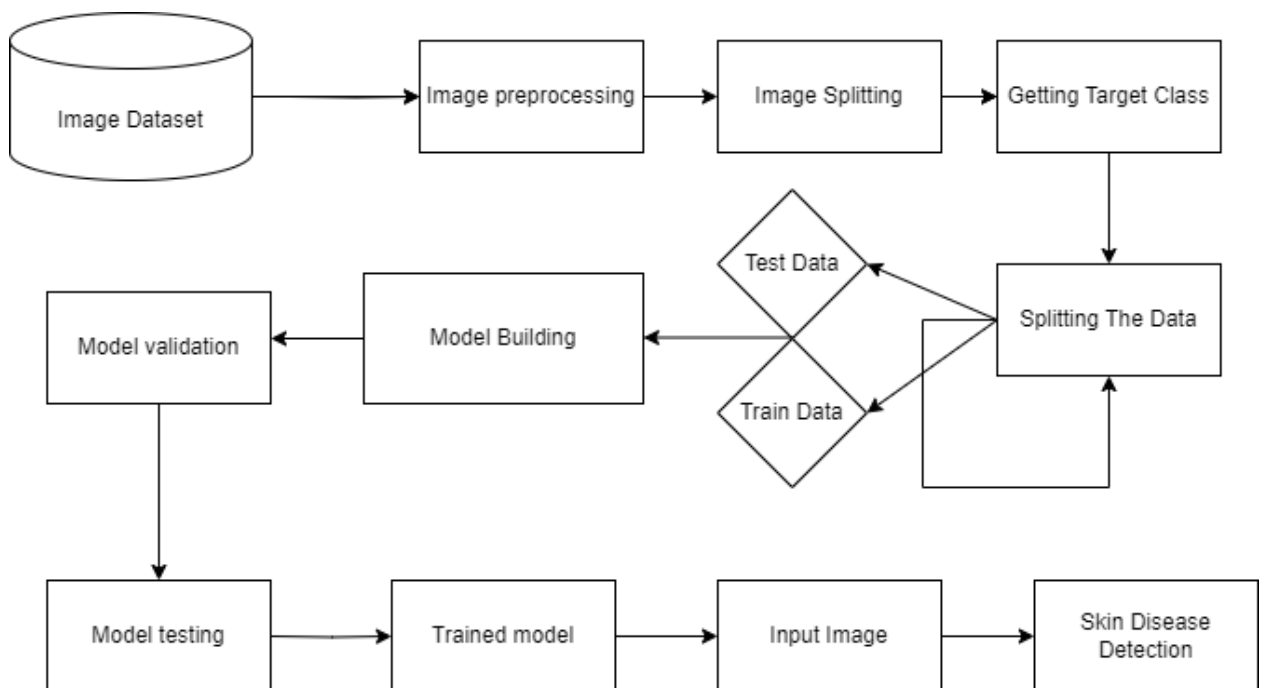


Fig 5.1 SYSTEM ARCHITECTURE

5.2 USE CASE DIAGRAM

A Use Case Diagram for a Skin Disease Identification System using Deep Learning Techniques would illustrate the key interactions and functionalities between actors and the system. The diagram would feature use cases like "Upload Images for Analysis", "Deep Learning Classification", "Automated Diagnosis Generation", and "Alert Healthcare Professionals." Actors may include patients, healthcare providers, data scientists, and system administrators. Each use case would outline the steps and interactions necessary to accurately detect and diagnose skin diseases through the application of deep learning algorithms. Visually representing these processes would highlight the efficiency and effectiveness of utilizing machine learning in healthcare diagnostics.

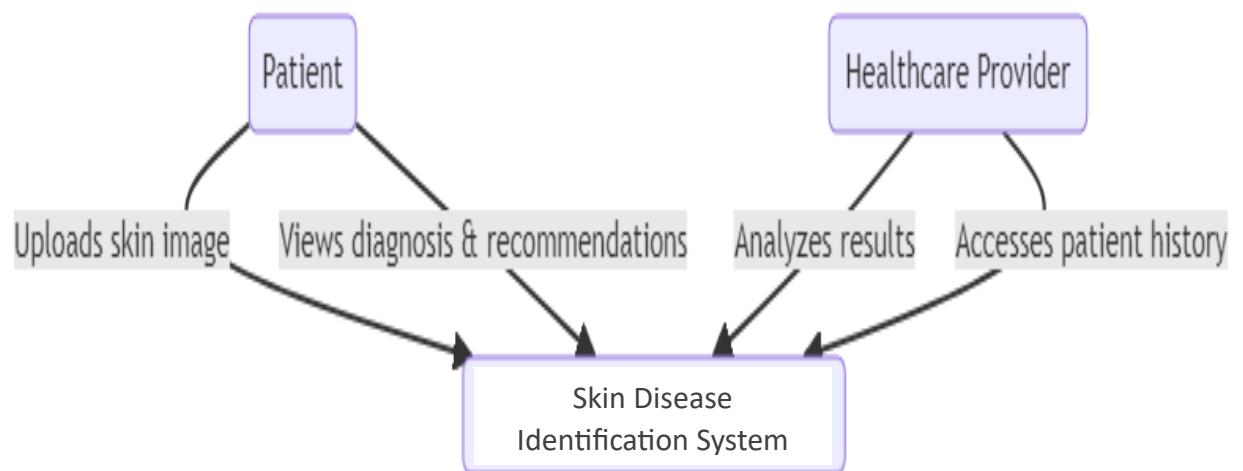


Fig 5.2 USE CASE DIAGRAM

5.3 ACTIVITY DIAGRAM

An activity diagram for Skin Disease Identification using deep learning would outline key stages such as data collection from images of skin lesions, image pre-processing for normalization, feature extraction to identify important patterns, model training using deep learning algorithms, prediction of the skin disease based on the learned patterns, and result analysis to evaluate the accuracy of the prediction. The diagram would illustrate the flow of information and decision-making processes, guiding stakeholders through the system's functionality and highlighting areas for optimization to enhance the accuracy of skin disease detection through machine learning techniques.

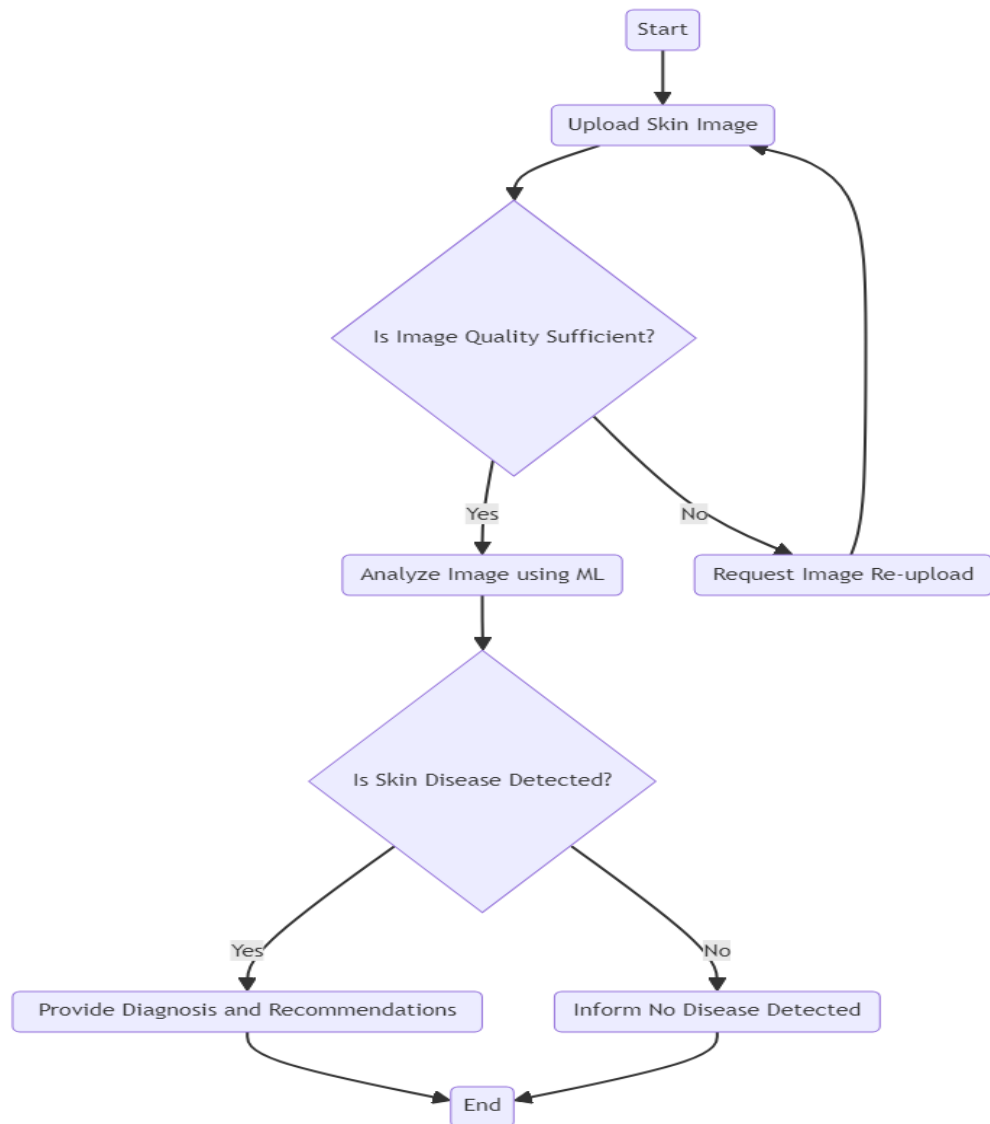


Fig 5.3 ACTIVITY DIAGRAM

5.4 SEQUENCE DIAGRAM

A sequence diagram for a Skin Disease Identification System utilizing Deep Learning Techniques would showcase the progression of interactions and activities among the system's components. The diagram would visually represent the sequential steps involved in the process, including data input, image processing for feature extraction, training of machine learning models, and generating prediction outputs. By delineating the flow of information and operations within the system, the sequence diagram aids in comprehending how machine learning algorithms analyze input data to identify and classify various skin diseases accurately.

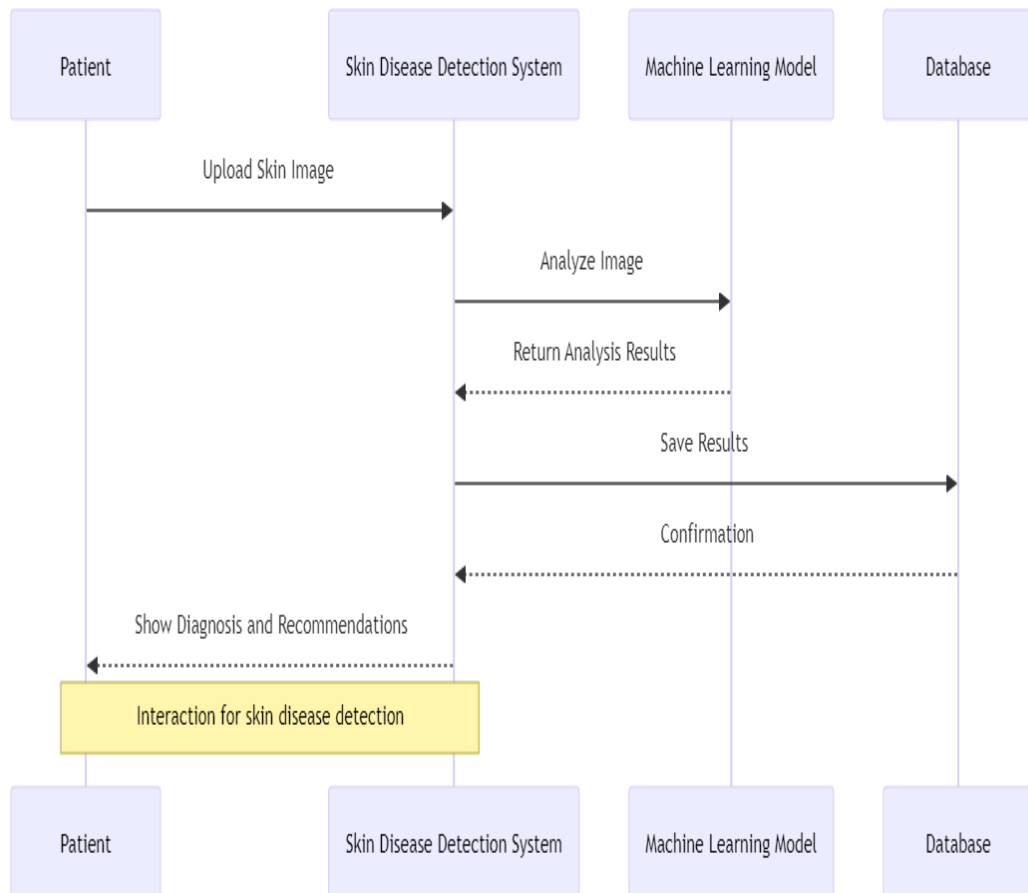


Fig 5.4 SEQUENCE DIAGRAM

4.5 CLASS DIAGRAM

A Class Diagram for a Skin Disease Identification System using Deep Learning techniques would include classes such as 'User', 'Image Input', 'Preprocessing Module', 'Feature Extraction Module', 'Deep Learning Model', 'Prediction Output', 'Alert System', 'Database', and 'Skin Disease Data'. Each class would have attributes and methods defining their functionalities within the system. Relationships between classes would illustrate how data flows between components for accurate detection and prediction of skin diseases. Overall, the diagram would showcase how different modules and entities collaborate to provide an efficient and reliable skin disease detection system leveraging Deep Learning algorithms.

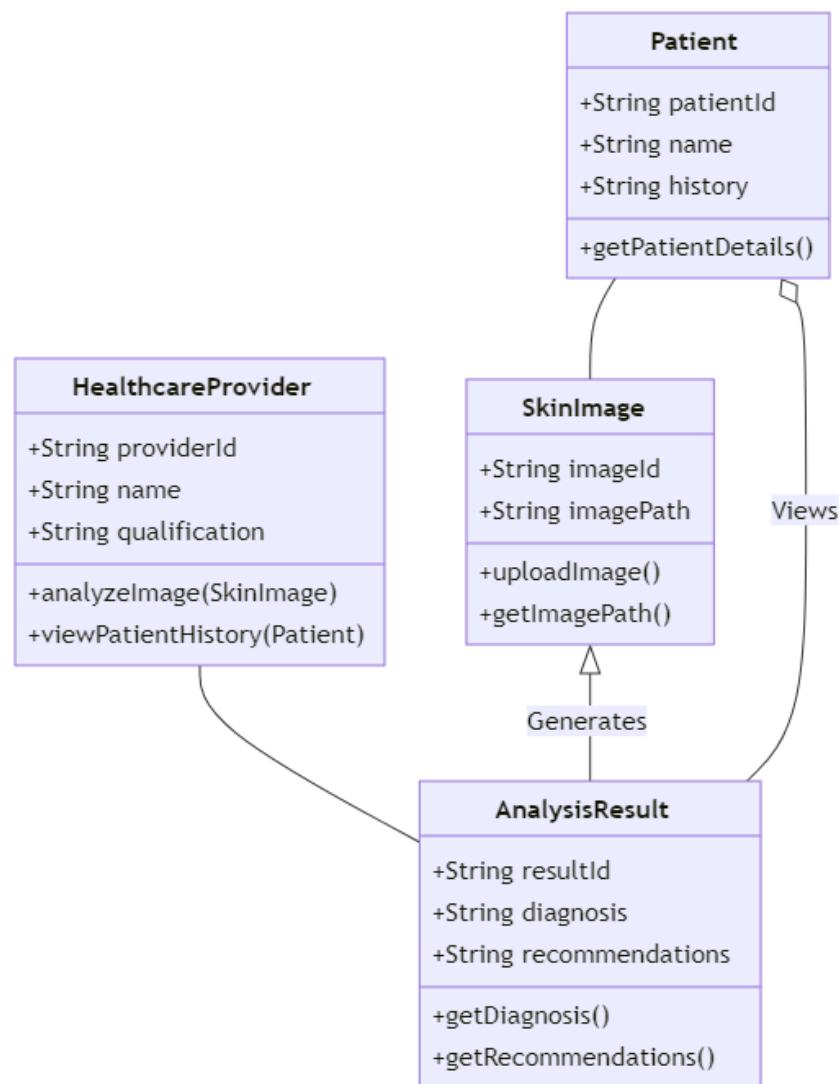


Fig 5.5 CLASS DIAGRAM

5.6 ER DIAGRAM

For a Skin Disease Identification system utilizing Deep Learning Techniques, the Entity Relationship (ER) diagram would showcase entities like skin images, diagnoses, treatments, patients, and medical professionals. The relationships between these entities would illustrate connections, such as a patient having multiple skin images or a diagnosis leading to a specific treatment plan. This diagram aids in structuring the database effectively for efficient storage and retrieval of data, supporting the accurate identification of skin conditions through machine learning algorithms. By visually mapping out these relationships, the ER diagram plays a crucial role in optimizing the system's performance in detecting and diagnosing skin diseases.

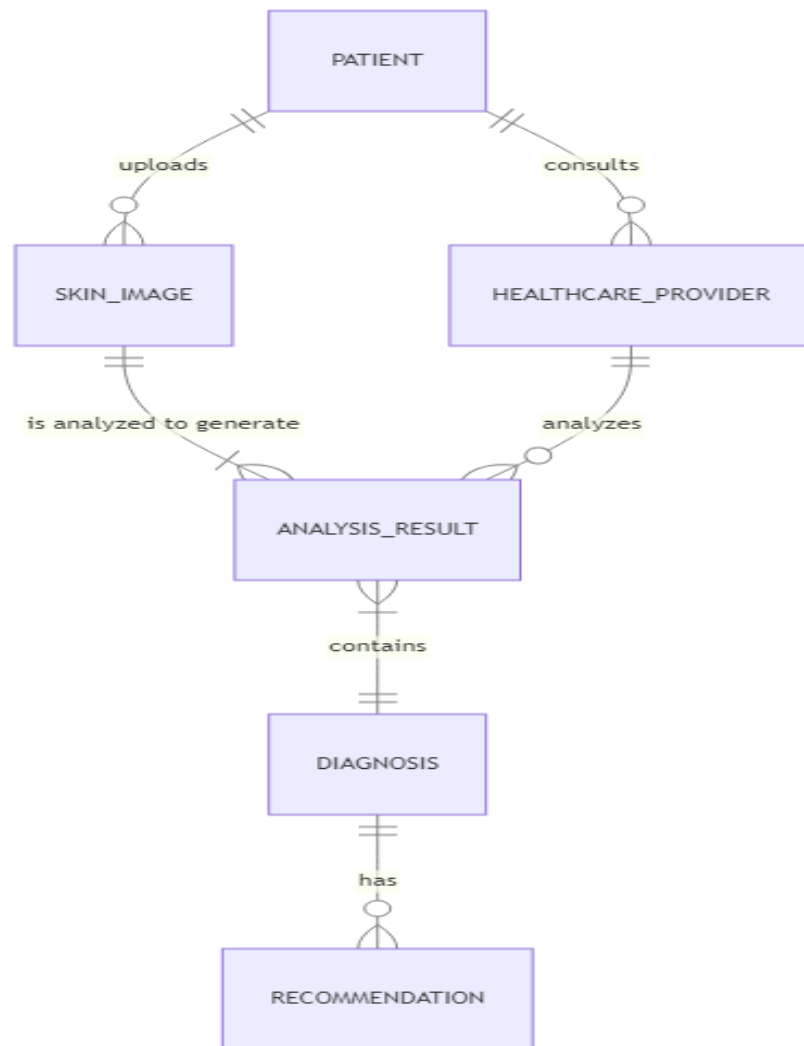


Fig 5.6 ER DIAGRAM

5.7 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a visual representation of how data moves through a system, showing inputs, processes, outputs, data storage, and external entities. It helps in understanding system functionality, spotting inefficiencies, and improving communication between technical and non-technical stakeholders. DFDs are vital in system analysis and design, ensuring accurate and efficient data processing. By using symbols to represent elements like processes and data flows, DFDs provide a clear overview of information movement, aiding in system optimization and effective decision-making.

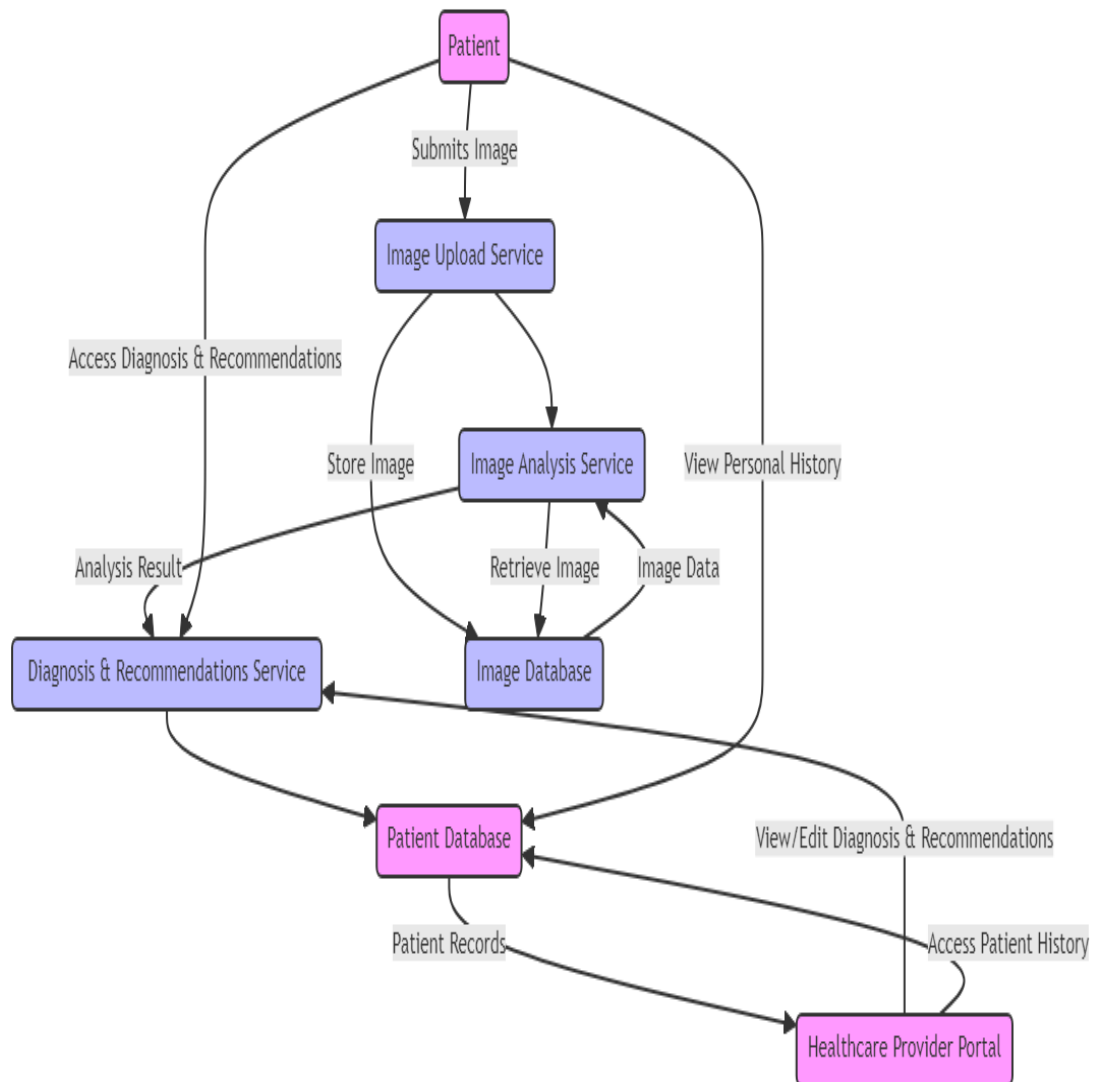


Fig 5.7 DATA FLOW DIAGRAM Level 1

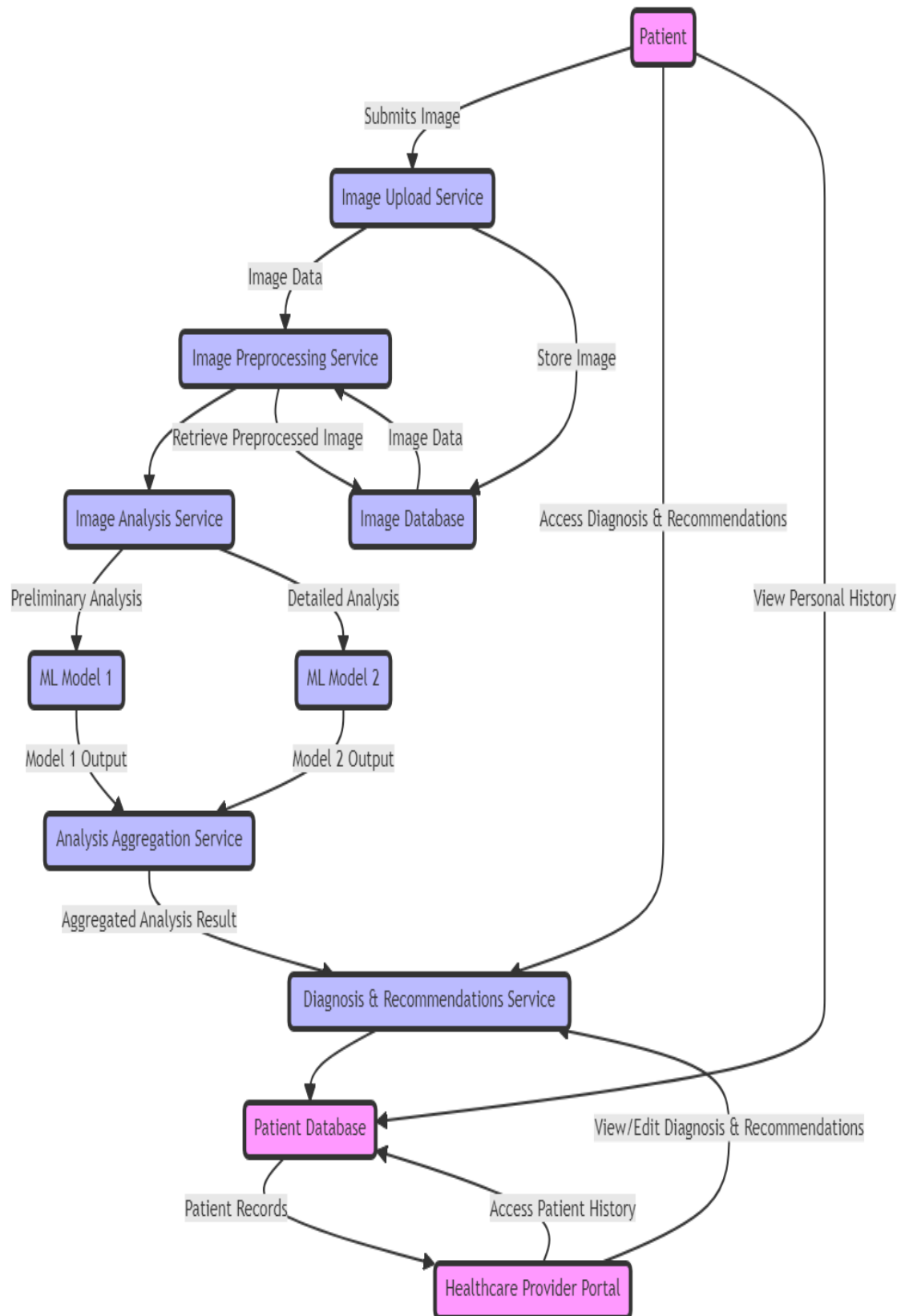


Fig 5.8 DATA FLOW DIAGRAM Level 2

CHAPTER VI

MODULE DESCRIPTION

CHAPTER 5

MODULE DESCRIPTION

6.1 IMAGE SOURCES

The Skin disease is the most common human malignancy, is primarily diagnosed visually, beginning with an initial clinical screening and followed potentially by dermoscopic analysis, a biopsy and histopathological examination. Automated classification of skin lesions using images is a challenging task owing to the fine-grained variability in the appearance of skin lesions. This the HAM10000 ("Human Against Machine with 10000 training images") dataset. It consists of 10015 dermoscopic images which are released as a training set for academic machine learning and deep learning purposes and are publicly available through the ISIC archive. This benchmark dataset can be used for machine learning and for comparisons with human experts. It has 7 different classes of skin cancer which are listed below:

1. Melanocytic nevi
2. Melanoma
3. Benign keratosis-like lesions
4. Basal cell carcinoma
5. Actinic keratoses
6. Vascular lesions
7. Dermatofibroma

6.2 DATA ANNOTATION

Expert Annotations: Engaging dermatologists and healthcare professionals to annotate the collected images with labels indicating the presence of specific skin diseases or conditions.

Ground Truth Labels: Ensuring the accuracy and reliability of annotations by cross-referencing them with diagnoses made by multiple dermatologists or through

histopathological confirmation for certain cases.

Metadata Collection: Recording relevant metadata for each image, including patient demographics (age, gender, ethnicity), lesion location, clinical history, contextual information.

6.3 ARCHITECTURE OF CNN

A Convolutional neural network (CNN) is one type of Artificial Neural Network. A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, segmentation and also for other auto correlated data.

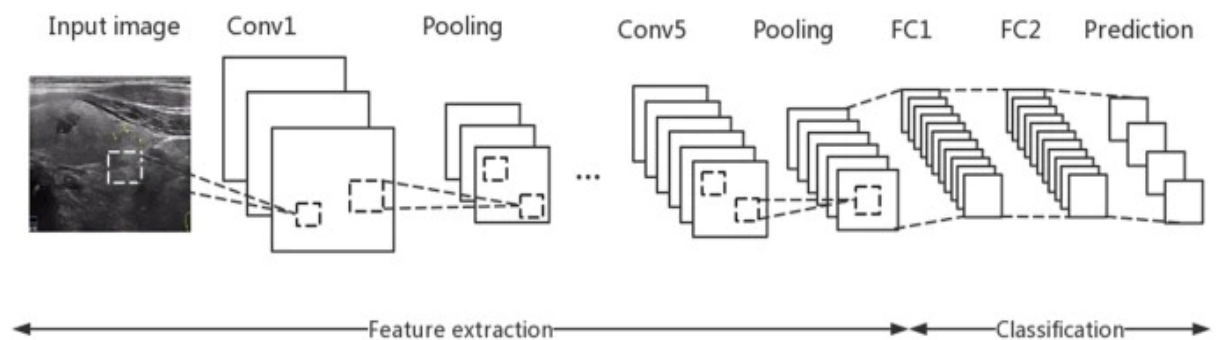


Fig. 6.1 Architecture of CNN

There are three ways to create Keras models:

➤ Sequential model

- The **Sequential model**, which is very straightforward (a simple list of layers), but is limited to single-input, single-output stacks of layers (as the name gives away).
- It allows us to create models layer by layer in sequential order. But it does not allow us to create models that have multiple inputs or outputs. It is best for simple stack of layers which have 1 input tensor and 1 output tensor.
- This model is not suited when any of the layer in the stack has multiple inputs or outputs. Even if we want non-linear topology, it is not suited.

➤ **Functional API**

- The **Functional API**, which is an easy-to-use, fully-featured API that supports arbitrary model architectures. For most people and most use cases, this is what you should be using. This is the Keras "industry strength" model.
- It provides more flexibility to define a model and add layers in keras. Functional API allows us to create models that have multiple input or output. It also allows us to share these layers. In other words. we can make graphs of layers using Keras functional API.
- As functional API is a data structure, it is easy to save it as a single file that helps in recreating the exact model without having the original code. Also its easy to model the graph here and access its nodes as well.

➤ **Model Subclassing**

- **Model subclassing**, where you implement everything from scratch on your own. Use this if you have complex, out-of-the-box research use cases.
- Sequential model does not allow you much flexibility to create your models. Functional API also only has a little of customization available for you. But you may create your own fully-customizable models in Keras. This is done by subclassing the Model class and implementing a call method. Input() is used to instantiate a Keras tensor.
- A Keras tensor is a symbolic tensor-like object, which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model.
- For instance, if `a`, `b` and `c` are Keras tensors, it becomes possible to do: `model = Model(input=[a, b], output=c)`

CNN KERNELS

Each convolutional layer contains a series of filters known as convolutional kernels. The filter is a matrix of integers that are used on a subset of the input pixel values, the same size as the kernel. Each pixel is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid cell, like a pixel, in the output channel/feature map. These are linear transformations; each convolution is a type of affine fn.

In computer vision the input is often a 3 channel RGB image. For simplicity, if we take a greyscale image that has one channel (a two-dimensional matrix) and a 3x3 convolutional kernel (a two-dimensional matrix). The kernel strides over the input matrix of numbers moving horizontally column by column, sliding/scanning over the first rows in the matrix containing the images pixel values. Then the kernel strides down vertically to subsequent rows. Note, the filter may stride over one or several pixels at a time, this is detailed further below.

In other non-vision applications, a one-dimensional convolution may slide vertically over an input matrix.

CNN PADDING:

To handle the edge pixels there are several approaches:

- Losing the edge pixels
- Padding with zero value pixels
- Reflection padding

Reflection padding is by far the best approach, where the number of pixels needed for the convolutional kernel to process the edge pixels are added onto the outside copying the pixels from the edge of the image. For a 3x3 kernel, one pixel needs to be added around the outside, for a 7x7 kernel then three pixels would be reflected around the outside. The pixels added around each side is the dimension, halved and rounded down.

Traditionally in many research papers, the edge pixels are just ignored, which loses a small proportion of the data and this gets increasing worse if there are many deep convolutional layers. For this reason, I could not find existing diagrams to easily convey some of the points here without being misleading and confusing stride 1 convolutions with stride 2 convolutions. With padding, the output from a input of width w and height h would be width w and height h (the same as the input with a single input channel), assuming the kernel takes a stride of one pixel at a time.

CNN STRIDES:

It is common to use a stride two convolution rather than a stride one convolution, where the convolutional kernel strides over 2 pixels at a time, for example our 3x3 kernel would start at position (1,1), then stride to (1,3), then to (1, 5) and so on, halving the size of the output channel/feature map, compared to the convolutional kernel taking strides of one.

With padding, the output from an input of width w , height h and depth 3 would be the ceiling of width $w/2$, height $h/2$ and depth 1, as the kernel outputs a single summed output from each stride.

6.4 SCALABILITY AND MAINTENANCE

Scalable Infrastructure: Implementing scalable data storage and management infrastructure capable of handling large volumes of images and metadata efficiently.

Versioning and Updates: Establishing protocols for versioning the dataset and incorporating new images, annotations, and metadata over time to keep the dataset up-to-date and relevant.

Community Engagement: Encouraging collaboration and contributions from the research community, healthcare providers, and other stakeholders to expand and enrich the dataset collaboratively.

6.5 IMAGE PREPROCESSING TECHNIQUES

Image preprocessing plays a crucial role in enhancing the quality of images and extracting relevant features for subsequent analysis in the context of skin disease identification using deep learning. Preprocessing techniques are employed to standardize images, remove noise, and enhance important features, thereby improving the performance of machine learning algorithms. Below are some commonly used images preprocessing techniques in this domain:

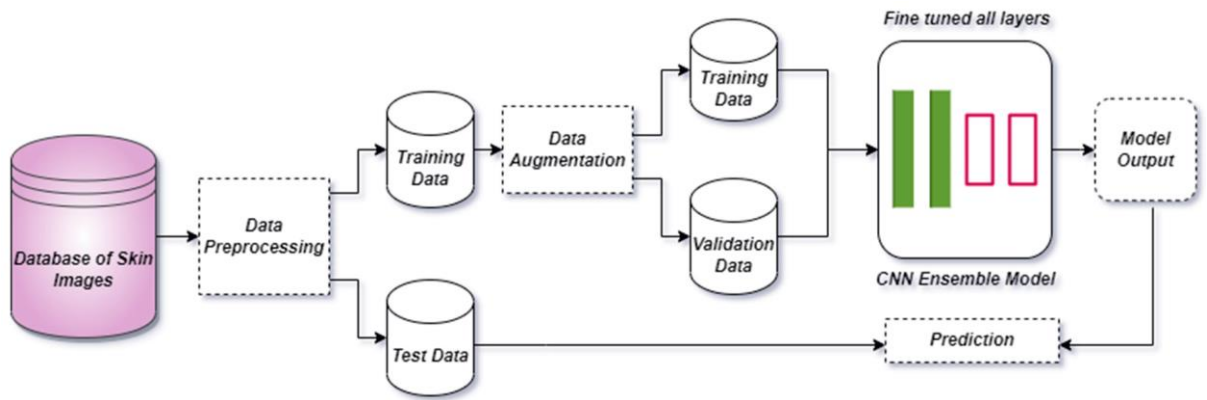


Fig. 6.2 Preprocessing Steps

Resizing and Rescaling: Images acquired from different sources may vary in size and resolution. Resizing and rescaling techniques are applied to ensure uniformity in image dimensions, facilitating consistent processing by deep learning models. Resizing reduces computational complexity and improves efficiency without significantly sacrificing image quality.

Normalization: Normalization is a technique used to standardize pixel values across images. By scaling pixel intensities to a common range (e.g., 0 to 1), normalization ensures that variations in lighting conditions and exposure levels do not affect the performance of deep learning algorithms. Normalization helps in reducing the impact of illumination changes and enhances the model's generalization capability.

Color Space Conversion: Skin disease diagnosis often relies on color information to identify characteristic features such as lesions, rashes, or discolorations. Converting images from the RGB color space to other color spaces such as HSV (Hue, Saturation, Value) or LAB (Luminance, A, B) can enhance the discriminative power of color features.

Different color spaces may emphasize different aspects of the image, making them more suitable for specific tasks such as lesion segmentation or feature extraction.

Noise Reduction: Images acquired from real-world settings may contain various types of noise, including Gaussian noise, salt-and-pepper noise, or motion blur. Noise reduction techniques such as median filtering, Gaussian blurring, or bilateral filtering are applied to suppress noise while preserving important details. Denoising improves the quality of images and enhances the performance of subsequent processing steps.

Contrast Enhancement: Contrast enhancement techniques are employed to improve the visibility of important features in images. Histogram equalization, adaptive histogram equalization, and contrast stretching are common methods used to adjust the distribution of pixel intensities, thereby enhancing image contrast and improving the visibility of subtle details. Contrast enhancement techniques can help in highlighting abnormalities and making them more discernible to deep learning algorithms.

Edge Detection: Edge detection algorithms are used to identify boundaries and contours of objects in images. Techniques such as Sobel, Prewitt, or Canny edge detection can highlight edges and boundaries of lesions or skin features, providing valuable information for subsequent analysis. Edge detection helps in segmenting regions of interest and extracting relevant features for classification and diagnosis.

Image Augmentation: Image augmentation techniques are employed to increase the diversity and variability of the training dataset. Techniques such as rotation, translation, flipping, or scaling are applied to generate augmented images, introducing variations in pose, orientation, and appearance. Image augmentation helps in improving the robustness and generalization capability of machine learning models, making them more effective in real-world scenarios.

6.6 DATA AUGMENTATION AND IMAGE ENHANCEMENT

In the context of skin disease identification using deep learning, data augmentation and image enhancement techniques are vital for increasing the diversity of the dataset, improving model generalization, and enhancing the quality of input images. These techniques play a crucial role in mitigating issues related to dataset scarcity, class imbalance, and variability in imaging conditions.

6.6.1 DATA AUGMENTATION

Data augmentation involves generating additional training samples by applying a variety of transformations to existing images. This technique helps in expanding the dataset, thereby improving the robustness and generalization ability of machine learning models. Common data augmentation techniques used in skin disease detection include:

Rotation: Rotating images by various angles (e.g., 90, 180, 270 degrees) introduces variations in pose and orientation, making the model more resilient to changes in image orientation.

Flip: Mirroring images horizontally or vertically introduces left-right or up-down reflections, increasing dataset variability and reducing bias towards specific orientations.

Scaling: Resizing images to different scales introduces variations in size, enabling the model to learn features at different resolutions and scales.

Translation: Shifting images horizontally or vertically introduces variations in position, simulating changes in camera angle or patient positioning.

Shearing: Applying shearing transformations to images introduces distortions along the x or y-axis, mimicking changes in perspective or camera angle.

Zooming: Cropping and resizing images to focus on specific regions of interest simulate variations in image magnification and field of view.

Noise Injection: Adding random noise to images simulates noise present in real-world imaging conditions, making the model more robust to noise artifacts.

6.6.2 IMAGE ENHANCEMENT

Image enhancement techniques are used to improve the quality and visibility of important features in skin images, enhancing the model's ability to detect and classify skin diseases accurately.

Contrast Enhancement: Adjusting the contrast of images using techniques such as histogram equalization or contrast stretching improves the visibility of subtle features, making lesions and abnormalities more discernible.

Brightness Adjustment: Modifying the brightness of images helps in compensating for variations in lighting conditions, ensuring consistent illumination across images.

Color Correction: Standardizing color characteristics across images using color calibration techniques ensures uniformity in color representation, reducing variability due to differences in imaging devices or conditions.

Sharpness Enhancement: Applying sharpening filters or deconvolution techniques enhances image sharpness, making fine details and textures more pronounced.

Artifact Removal: Removing artifacts such as dust, scratches, or motion blur from images improves image quality and reduces interference with disease detection.

Edge Enhancement: Emphasizing edges and boundaries of lesions using edge detection algorithms improves segmentation accuracy and facilitates feature extraction.

Resolution Enhancement: Up sampling low-resolution images using techniques such as super-resolution enhances image clarity and detail, improving the model's ability to identify subtle features.

6.7 FEATURE EXTRACTION AND SELECTION

6.7.1 FEATURE EXTRACTION

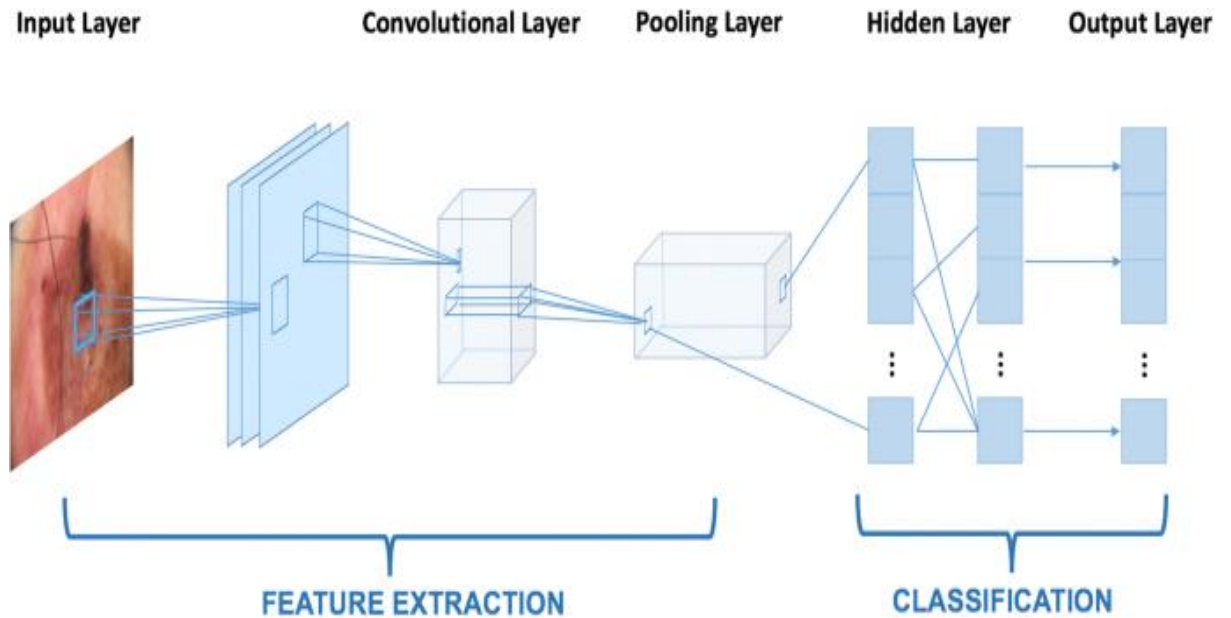


Fig .6.3 Feature Extraction

Feature extraction and selection are critical steps in the process of skin disease identification using deep learning. These steps involve identifying and extracting informative features from input images to represent distinct characteristics of skin lesions or diseases. Effective feature extraction and selection methods play a key role in improving the discriminative power of machine learning models and facilitating accurate classification of skin conditions

Feature Extraction: Feature extraction involves transforming raw input data (such as images) into a set of relevant features that capture distinctive patterns or characteristics of skin diseases. In the context of skin disease detection, features can include texture descriptors, color histograms, shape properties, and other quantitative measurements. Common feature extraction techniques used in this domain include:

Texture Analysis: Texture features capture spatial patterns and variations in pixel intensities within skin lesions. Techniques such as gray-level co-occurrence matrix (GLCM), local binary patterns (LBP), and Gabor filters are used to extract texture

descriptors that characterize textural properties such as smoothness, roughness, or granularity.

Color Descriptors: Color features represent the distribution of color intensities within skin lesions. Color histograms, color moments, and color spaces (e.g., RGB, HSV, LAB) are used to extract color-based descriptors that capture information about hue, saturation, and brightness. Color features are particularly useful for distinguishing between different types of lesions based on their pigmentation or vascularization.

Shape Analysis: Shape features quantify the geometric properties and spatial arrangement of lesions. Techniques such as contour-based analysis, Fourier descriptors, and shape context descriptors are used to extract shape-based features that characterize the shape, size, and symmetry of lesions. Shape features are valuable for distinguishing between lesions with distinct morphological characteristics, such as nodules, patches, or ulcers.

Intensity Statistics: Intensity-based features capture statistical properties of pixel intensities within lesions. Mean, standard deviation, skewness, and kurtosis are common intensity statistics used to quantify the distribution and variability of pixel intensities. Intensity features provide information about the overall brightness and contrast of lesions, aiding in their characterization and classification.

Spatial Relationships: Spatial features describe the spatial arrangement and interactions between different regions within lesions. Spatial relationships such as adjacency, connectivity, and spatial distribution are captured using techniques such as spatial histograms or spatial autocorrelation. Spatial features are useful for capturing spatial patterns and configurations of lesions, providing additional discriminative information for classification.

6.7.2 FEATURE SELECTION

Feature Selection: Feature selection aims to identify the most relevant and discriminative features from the extracted feature set, eliminating redundant or irrelevant features that may degrade model performance or increase computational complexity. Effective feature selection methods help in improving model interpretability, reducing overfitting, and enhancing classification accuracy.

Filter Methods: Filter methods evaluate individual features based on their statistical properties (e.g., correlation, mutual information) or relevance to the target variable (e.g., chi-square test, ANOVA). Features are ranked or scored according to their importance, and a subset of the most informative features is selected for model training.

Wrapper Methods: Wrapper methods evaluate feature subsets by training and testing machine learning models on different combinations of features. Techniques such as forward selection, backward elimination, and recursive feature elimination (RFE) iteratively select features that optimize model performance. Wrapper methods consider the interaction between features and directly optimize model performance metrics, but they are computationally intensive and may be prone to overfitting.

Embedded Methods: Embedded methods integrate feature selection into the model training process, allowing the model to automatically learn feature importance during training. Techniques such as LASSO (Least Absolute Shrinkage and Selection Operator) regularization, decision tree pruning, and feature importance ranking in ensemble methods (e.g., Random Forest, Gradient Boosting) are examples of embedded feature selection methods. Embedded methods are computationally efficient and often result in sparse feature representations, making them suitable for high-dimensional datasets.

Dimensionality Reduction: Dimensionality reduction techniques such as Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) project high-dimensional feature vectors into lower-dimensional representations while preserving important structural information. Dimensionality reduction helps in reducing the computational complexity of models and visualizing high-dimensional data in lower-dimensional spaces.

6.8 DATA NORMALIZATION AND SPLITTING

6.8.1 DATA NORMALIZATION

In the context of skin disease detection using machine learning, data normalization and splitting are essential preprocessing steps that ensure the quality and reliability of the dataset and facilitate the training, validation, and evaluation of machine learning models. These steps involve standardizing input data and partitioning it into separate sets for training, validation, and testing. Below are detailed explanations of data normalization and splitting techniques:

Data Normalization: Data normalization is the process of scaling input features to a standard range or distribution, ensuring that all features contribute equally to model training and preventing biases due to differences in scale or magnitude. In skin disease detection, normalization is particularly important for image data, where pixel values may vary widely across images due to differences in lighting conditions, exposure levels, or imaging devices. Common normalization techniques include:

1. Min-Max Scaling: Rescaling feature values to a fixed range, typically between 0 and 1, using the formula:

$$[X_{((norm))} = \text{frac}(X - X_{((min))})(X_{((max))} - X_{((min))})]$$

Where,

- (X) is the original feature value, $(X_{((min))})$ is the minimum value of the feature, and $(X_{((max))})$ is the maximum value of the feature.
- Min-max scaling preserves the original distribution of feature values while ensuring that they fall within a standardized range.

2. Z-Score Normalization (Standardization): Standardizing feature values to have a mean of 0 and a standard deviation of 1, using the formula:

$$[X_{((norm))} = \text{frac}(X - \mu)(\sigma)]$$

Where,

- (X) is the original feature value, (μ) is the mean of the feature, and (σ) is

the standard deviation of the feature. Z-score normalization transforms feature values to a standard Gaussian distribution, making them more comparable and interpretable.

3. Robust Scaling: Scaling feature values to be robust to outliers by subtracting the median and dividing by the interquartile range (IQR), using the formula:

$$[X_{((norm))} = \frac{X - (\text{median})(X)}{(IQR)(X)}]$$

Robust scaling is particularly useful when the dataset contains outliers or extreme values that may skew the normalization process.

Data normalization ensures that input features have consistent scales and distributions, facilitating convergence during model training and preventing numerical instabilities. Normalized data also improves the performance of optimization algorithms and enhances the interpretability of model parameters.

6.8.2 DATA SPLITTING

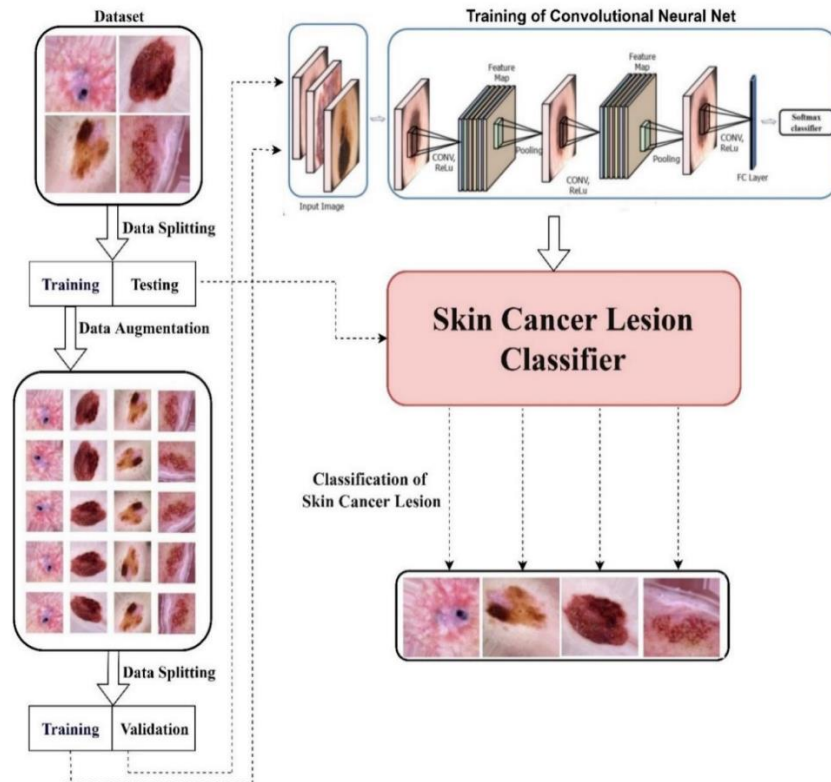


Fig .6.4 Data Splitting

Data splitting involves partitioning the dataset into separate subsets for training, validation, and testing, enabling the evaluation of model performance on unseen data and preventing overfitting. Common data splitting strategies include:

1. Training Set: The training set comprises a majority portion of the dataset used to train the machine learning model. Training data are used to optimize model parameters and learn patterns from input features.

2. Validation Set: The validation set is used to tune hyperparameters and evaluate model performance during training. It provides an unbiased estimate of model performance on unseen data and helps in preventing overfitting by monitoring performance on a separate dataset.

3. Test Set: The test set is a completely independent subset of the dataset used to evaluate the final performance of the trained model. Test data are not used during model training or hyperparameter tuning and provide an objective assessment of model generalization to unseen data.

Common data splitting ratios include the 70-15-15 split (70% training, 15% validation, 15% testing) or the 80-10-10 split (80% training, 10% validation, 10% testing). Stratified sampling may be used to ensure that each subset preserves the class distribution of the original dataset, especially in cases of imbalanced datasets.

CHAPTER VII

MODEL

IMPLEMENTATION

CHAPTER 7

MODEL IMPLEMENTATION

7.1 REQUIREMENT ANALYSIS AND SPECIFICATION

Requirement analysis consists of the tasks to determine the requirements and conditions to get the product by considering all possible user needs. The analysis on performing the task involves two requirements likely input data and output data. These requirements are explained in the following section.

7.1.1 INPUT DATA

The CNN model expects the preprocessed images as input during training, validation, and testing. The input shape of the CNN model should match the dimensions of the preprocessed images. Adjust the input layer of the CNN architecture accordingly to accommodate the chosen image size and number of color channels.

7.1.2 OUTPUT DATA

A detector model was made using the Convolution Neural Network (CNN), which takes the images as input and process the data of the images and will predict the results as what kind or classification of skin disease. As the various types are Melanocytic nevi, Melanoma, Benign keratosis-like lesions, Basal cell carcinoma, Actinic keratoses, Vascular lesions, Dermatofibroma Convolution Neural Network (CNN) are the best way to predict the diseases.

7.2 PROJECT REQUIREMENTS

Requirements are the basic constraints that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Functional requirements
2. Non-Functional requirements
3. Environment requirements
 - A. Hardware requirements
 - B. software requirements

7.2.1 FUNCTIONAL REQUIREMENTS

The software requirements specification is a technical specification of requirements for the software product. It is the first step in the requirements analysis process. It lists requirements of a particular software system. The following details to follow the special libraries like TensorFlow, keras, matplotlib, seaborn, pandas etc.

7.2.2 NON-FUNCTIONAL REQUIREMENTS

Process of functional steps,

1. Problem defines
2. Preparing data
3. Evaluating algorithm
4. Improving results
5. Prediction the result

7.3 PROJECT WORKFLOW:

- Load the data
 - Loading the given dataset
 - Import required libraries packages
- **Pre-process the data**
 - Reshape, data augmentations
- **Define model**
 - Sequential or Functional
 - Number of layers to be used, Number of nodes to be used in the model, Evaluation metrics
- **Compile the model**
 - Define loss function, optimizer, weights and bias
- **Fit the model**
 - Train data, Test data, epoch, Batch size.

7.4 STEPS INVOLVED IN SKIN DISEASES IDENTIFICATION

7.4.1 DATA COLLECTION

The Kaggle datasets that we are employing in this model's creation include the data of 10015 image records of features extracted. These data sets include types of skin diseases images of human.

7.4.2 PREPARING DATA

In this process, we will load the data, extract features from it, then split the dataset into training, validation and testing sets as shown in Fig.6.1. Then, we'll initialize CNN model and train the model. Finally, we'll calculate the accuracy of our model.

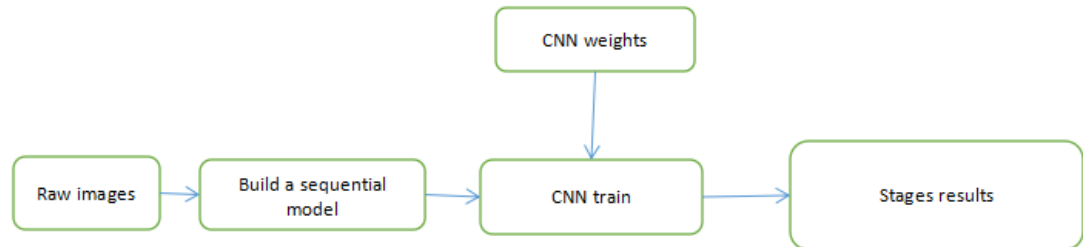


Fig. 7.1 Flow of Pre-processing and Training the CNN Model

7.4.3 MAKING PREDICTION

The CNN's inception layer receives the human emotions after being trimmed and removed. Utilizing a convolution neural network, the training process entails feature extraction and classification as shown in Fig.6.2.

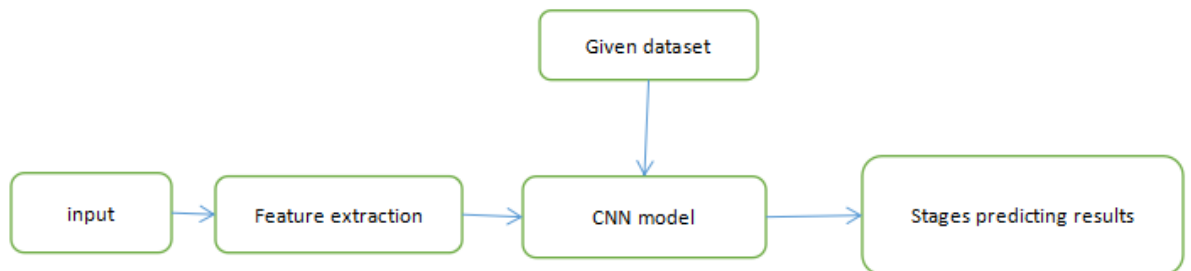


Fig.7.2 Flowchart for the Emotion Identification

7.4.4 BUILDING MODEL

7.4.4.1 SPLITTING DATASET

The data use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. It has the test dataset (or subset) in order to test our models and it will do this using Tensor flow library in Python using the Keras method.

Deep learning needs data gathering have lot of past images data. Training and testing this model working and predicting correctly as shown in Fig.5.3.

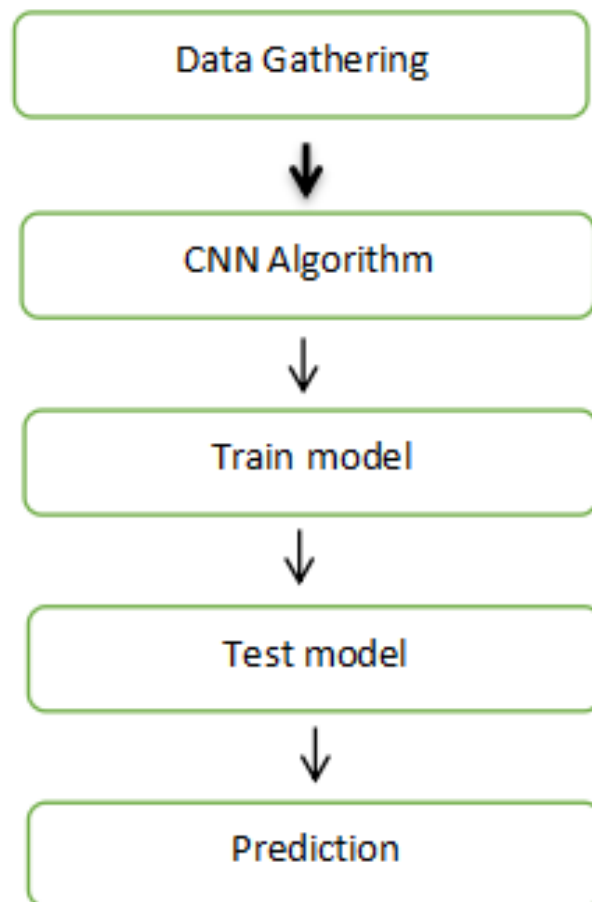


Fig.7.3 Construction of Detecting Mode

7.4.4.2 CONVOLUTIONAL NEURAL NETWORK:

A Convolutional neural network (CNN) is one type of Artificial Neural Network. A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, segmentation and also for other auto correlated data are depicted in Fig.5.4.

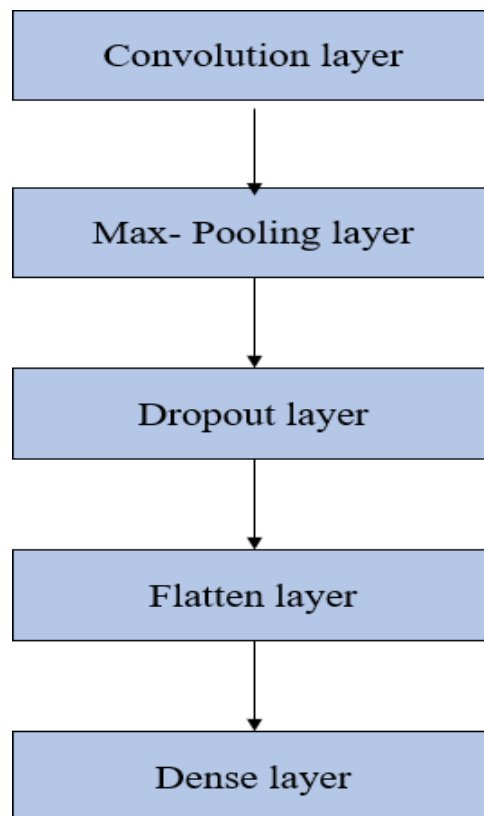


Fig.7.4 Flow of CNN Model

➤ DENSE LAYER

Dense implements the operation: $\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$ where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True). These are all attributes of Dense.

Note: If the input to the layer has a rank greater than 2, then Dense computes

the dot product between the inputs and the kernel along the last axis of the inputs and axis 0 of the kernel (using `tf.tensordot`). For example, if input has dimensions `(batch_size, d0, d1)`, then we create a kernel with shape `(d1, units)`, and the kernel operates along axis 2 of the input, on every sub-tensor of shape `(1, 1, d1)` (there are `batch_size * d0` such sub-tensors). The output in this case will have shape `(batch_size, d0, units)`

Besides, layer attributes cannot be modified after the layer has been called once (except the trainable attribute). When a popular kwarg `input_shape` is passed, then keras will create an input layer to insert before the current layer. This can be treated equivalent to explicitly defining an `InputLayer`.

Arguments:

- **units:** Positive integer, dimensionality of the output space.
- **activation:** Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).
- **use_bias:** Boolean, whether the layer uses a bias vector.
- **kernel_initializer:** Initializer for the kernel weights matrix.
- **bias_initializer:** Initializer for the bias vector.
- **kernel_regularizer:** Regularizer function applied to the kernel weights matrix.
- **bias_regularizer:** Regularizer function applied to the bias vector.
- **activity_regularizer:** Regularizer function applied to the output of the layer (its "activation").
- **kernel_constraint:** Constraint function applied to the kernel weights matrix.
- **bias_constraint:** Constraint function applied to the bias vector.

Input shape:

N-D tensor with shape: `(batch_size, ..., input_dim)`. The most

common situation would be a 2D input with shape (batch_size, input_dim).

Output shape:

N-D tensor with shape: (batch_size, ..., units). For instance, for a 2D input with shape (batch_size, input_dim), the output would have shape (batch_size, units).

➤ **DROPOUT LAYER**

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by $1/(1 - \text{rate})$ such that the sum over all inputs is unchanged.

Note that the Dropout layer only applies when training is set to True such that no values are dropped during inference. When using `model.fit`, training will be appropriately set to True automatically, and in other contexts, you set the kwarg explicitly to True when calling the layer.

Arguments:

- **rate:** Float between 0 and 1. Fraction of the input units to drop.
- **noise_shape:** 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch_size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use `noise_shape=(batch_size, 1, features)`.
- **seed:** A Python integer to use as random seed.

➤ **FLATTEN LAYER**

It is used to flatten the dimensions of the image obtained after convolving it. Dense: It is used to make this a fully connected model and is the hidden layer. Dropout: It is used to avoid over fitting on the dataset and dense is the output layer contains only one neuron which decide to which category image belongs.

Flatten is used to flatten the input. For example, if flatten is applied to layer having input shape as (batch_size, 2,2), then the output shape of the layer will be (batch_size, 4)

Flatten has one argument as follows

```
keras.layers.Flatten(data_format =  
None)
```

data_format is an optional argument and it is used to preserve weight ordering when switching from one data format to another data format. It accepts either channels_last or channels_first as value. channels_last is the default one and it identifies the input shape as (batch_size, ..., channels) whereas channels_first identifies the input shape as (batch_size, channels, ...)

➤ IMAGE DATA GENERATOR

It is that rescales the image, applies shear in some range, zooms the image and does horizontal flipping with the image. This Image Data Generator includes all possible orientation of the image.

➤ TRAINING PROCESS

train_datagen.flow_from_directory is the function that is used to prepare data from the train_dataset directory Target_size specifies the target size of the image. Test_datagen.flow_from_directory is used to prepare test data for the model and all is similar as above. fit_generator is used to fit the data into the model made above, other factors used are steps_per_epochs tell us about the number of times the model will execute for the training data.

➤ EPOCHS

It tells us the number of times model will be trained in forward and backward pass.

➤ VALIDATION PROCESS

Validation_data is used to feed the validation/test data into the model.

Validation_steps denote the number of validation/test samples.

7.5 MODEL SELECTION

1.Algorithm Selection: The choice of deep learning algorithm depends on factors such as the nature of the problem, the size and complexity of the dataset, and computational resources. Commonly used algorithms for image classification tasks include convolutional neural networks (CNNs), support vector machines (SVMs), decision trees, and ensemble methods like Random Forests or Gradient Boosting Machines.

2.Architecture Design: For skin disease detection, convolutional neural networks (CNNs) are widely used due to their ability to automatically learn hierarchical features from raw image data. The architecture of the CNN, including the number of layers, filter sizes, and activation functions, is customized based on the specific requirements of the task and the characteristics of the dataset. Transfer learning techniques, such as fine-tuning pre-trained CNN models (e.g., VGG, ResNet, Inception), can be employed to leverage the features learned from large-scale image datasets like ImageNet.

3.Hyperparameter Tuning: Hyperparameters, such as learning rate, batch size, dropout rate, and optimization algorithm, significantly impact the performance of the model. Hyperparameter tuning techniques, including grid search, random search, or Bayesian optimization, are used to systematically search for the optimal combination of hyperparameters that maximize model performance on a validation set.

4.Validation Strategy: Cross-validation or holdout validation is used to assess model performance and generalizeability. Cross-validation partitions the dataset into multiple subsets (folds), with each fold used for training and validation in turn. Holdout validation splits the dataset into training and validation sets, with the model trained on the training set and evaluated on the validation set. The validation strategy helps in selecting the best-performing model and detecting overfitting.

7.6 TRAINING PROCESS AND TUNING PARAMETERS

The training process and tuning parameters are crucial components in developing a deep learning model for skin disease detection. This involves iteratively adjusting model hyperparameters to optimize performance and training the model using appropriate techniques. Below is a detailed explanation of the training process and parameter tuning.

7.6.1 TRAINING PROCESS

1.Data Preparation: Preprocess the input data by resizing, normalizing, and augmenting images to ensure uniformity and enhance model generalization. Divide the dataset into training, validation, and test sets according to the chosen validation strategy.

2.Model Construction: Implement the selected model architecture using deep learning frameworks like TensorFlow, PyTorch, or Keras. Define the layers, activations, and loss functions of the model. Incorporate techniques such as batch normalization, dropout, and data augmentation to improve model robustness and prevent overfitting.

3.Hyperparameter Initialization: Initialize the model's hyperparameters, including learning rate, batch size, number of epochs, optimizer (e.g., SGD, Adam), weight initialization method, and regularization strength. The initial values of these hyperparameters significantly influence the convergence and performance of the model during training.

4.Training Loop: Iterate through the training data in mini-batches and update model parameters using backpropagation and gradient descent optimization. Monitor training progress by evaluating performance metrics (e.g., accuracy, loss) on the validation set. Adjust hyperparameters iteratively based on validation performance.

5.Early Stopping: Implement early stopping to prevent overfitting and improve training efficiency. Monitor the validation loss during training, and stop training when the validation loss begins to increase, indicating that the model is overfitting to the training data.

6.Model Evaluation: After training completes, evaluate the trained model on the test set to assess its performance on unseen data. Compute evaluation metrics such as accuracy, precision, recall, and F1-score to measure classification performance.

7.6.2 PARAMETER TUNING

1.Learning Rate: The learning rate determines the step size of parameter updates during optimization. Choose an appropriate learning rate that balances convergence speed and stability. Use techniques like learning rate schedules, adaptive learning rates (e.g., Adam), or learning rate annealing to adjust the learning rate during training.

2.Batch Size: The batch size specifies the number of samples processed in each training iteration. Larger batch sizes lead to faster training but may result in reduced generalization performance and increased memory usage. Experiment with different batch sizes to find the optimal balance between training speed and model performance.

3.Number of Epochs: An epoch refers to one complete pass through the entire training dataset. Train the model for a sufficient number of epochs to allow it to converge to an optimal solution without overfitting. Use early stopping to prevent training for too many epochs and avoid overfitting.

4.Optimizer: Choose an appropriate optimization algorithm (e.g., SGD, Adam, RMSprop) and its associated parameters (e.g., momentum, decay rates) based on the characteristics of the dataset and model architecture. Experiment with different optimizers and parameter settings to find the combination that yields the best performance.

5.Regularization: Regularization techniques such as L1 and L2 regularization, dropout, and batch normalization help prevent overfitting by imposing constraints on model parameters or introducing noise during training. Tune regularization strengths and dropout rates to control model complexity and improve generalization performance.

6.Weight Initialization: Initialize model weights using appropriate techniques (e.g., Xavier, He initialization) to ensure stable and efficient training. Experiment with different initialization methods to prevent vanishing or exploding gradients and facilitate convergence.

7.Architecture Modifications: Adjust the model architecture by adding or removing layers, changing layer sizes, or incorporating additional features (e.g., attention mechanisms) to improve performance. Experiment with different architectural modifications to find the optimal model configuration for the task.

7.7 EVALUATION METRICS AND PERFORMANCE ANALYSIS

Four evaluation metrics and performance analyses commonly used for skin disease identification using deep learning include accuracy, sensitivity, specificity, and F1 score. Accuracy measures the overall correctness of the model predictions, while sensitivity evaluates the ability of the model to correctly identify positive cases of skin disease. Specificity, on the other hand, measures the model's ability to correctly identify negative cases.

The F1 score combines both precision and recall to provide a balanced evaluation of the model's performance. High accuracy, sensitivity, and specificity levels indicate a reliable model for skin disease detection, while a high F1 score suggests a balanced trade-off between precision and recall. These metrics are essential in assessing the effectiveness and reliability of deep learning algorithms for skin disease detection, guiding researchers and practitioners in optimizing model performance.

CHAPTER VIII

SYSTEM TESTING

CHAPTER 8

SYSTEM TESTING

Discovering and fixing such problems is what testing is all about. The purpose of testing is to find and correct any problems with the final product. It's a method for evaluating the quality of the operation of anything from a whole product to a single component. The goal of stress testing software is to verify that it retains its original functionality under extreme circumstances. There are several different tests from which to pick. Many tests are available since there is such a vast range of assessment options.

Who Performs the Testing: All individuals who play an integral role in the software development process are responsible for performing the testing. Testing the software is the responsibility of a wide variety of specialists, including the End Users, Project Manager, Software Tester, and Software Developer.

When it is recommended that testing begin: Testing the software is the initial step in the process. begins with the phase of requirement collecting, also known as the Planning phase, and ends with the stage known as the Deployment phase. In the waterfall model, the phase of testing is where testing is explicitly arranged and carried out. Testing in the incremental model is carried out at the conclusion of each increment or iteration, and the entire application is examined in the final test.

When it is appropriate to halt testing: Testing the programme is an ongoing activity that will never end. Without first putting the software through its paces, it is impossible for anyone to guarantee that it is completely devoid of errors. Because the domain to which the input belongs is so expansive, we are unable to check every single input.

8.1 TYPES OF TESTING

8.1.1 UNIT TESTING

Skin Disease identification using deep learning involves training models to accurately classify skin conditions based on visual images. Unit testing plays a vital role in ensuring the performance and reliability of these models. Here are three test cases to validate the system.

Testcase1: Input Data Verification

- **Purpose:** Validate that input images are processed correctly by the machine learning model.
- **Steps:** Provide sample skin images as input and check if the model correctly classifies them.
- **Expected Outcome:** Model accurately identifies and categorizes different skin diseases.

Testcase2: Boundary Test

- **Purpose:** Evaluate the model's performance with extreme or challenging cases.
- **Steps:** Feed images with rare or severe skin conditions to assess the model's response.
- **Expected Outcome:** The model should still provide meaningful results for uncommon Skin diseases.

Testcase3: Integration Test

- **Purpose:** Ensure smooth interaction between data preprocessing, model training, and prediction task.
- **Steps:** Verify that all components work together seamlessly to detect skin diseases accurately.
- **Expected Outcome:** Components collaborate effectively to deliver reliable skin disease classification results.

8.1.2 INTEGRATION TESTING

For Skin Disease identification using deep learning, integration testing plays a crucial role in ensuring the seamless functioning of various system components.

Testcase1:

Verify the integration between the image acquisition module and the image preprocessing algorithm to ensure accurate and consistent image input for disease detection.

Testcase2:

Conduct a test to confirm the connection between the machine learning model and the image classification output to accurately identify different skin diseases based on input images.

Testcase3:

Test the integration of the diagnosis results with the user interface to display detailed information about detected skin diseases and recommended treatments effectively.

These test cases aim to validate the integration of key components within the system for precise and reliable skin disease identification using deep learning techniques.

8.1.3 FUNCTIONAL TESTING

For Skin Disease identification using deep learning, functional testing is crucial to ensure the accuracy and reliability of the system.

Testcase1: Data Collection

Verify that the system can gather various types of skin images from different sources and ensure they are stored correctly for processing. The test should validate that the images are of good quality and diverse enough to represent different skin conditions accurately.

Testcase2: Model Training

Confirm that the machine learning models are trained effectively on the collected dataset to accurately classify different skin diseases. This test should evaluate the model's ability to generalize well on unseen data and adapt to new skin conditions.

Testcase3: Prediction Accuracy

Evaluate the system's accuracy in detecting and classifying skin diseases in real-time scenarios. Test the precision, recall, and F1 score of the system's predictions to ensure it provides reliable results for clinical use.

By conducting these three test cases, we can ensure that the Skin Disease detection system using machine learning performs effectively and accurately assists in diagnosing various skin conditions.

CHAPTER IX

MODEL DEPLOYMENT

CHAPTER 9

MODEL DEPLOYMENT

9.1 IMPORT THE GIVEN IMAGE FROM DATASET

We have to import our data set using keras preprocessing image data generator function also we create size, rescale, range, zoom range, horizontal flip. Then we import our image dataset from folder through the data generatorfunction. Here we set train, test, and validation also we set target size, batch sizeand class-mode from this function we have to train using our own created networkby adding layers of CNN.

9.2 DEPLOYING THE MODEL IN STREAMLIT FRAMEWORK AND HTML

In this module the trained deep learning model is converted into hierarchical data format file (.h5 file) which is then deployed in our streamlit framework for providing better user interface and predicting the output whether the given image is Melanocytic nevi, Melanoma, Benign keratosis-like lesions, Basal cell carcinoma, Actinic keratoses, Vascular lesions, Dermatofibroma.

In the domain of skin disease identification using deep learning, the process of data acquisition and preprocessing plays a pivotal role in ensuring the accuracy and efficiency of the model. Data acquisition involves gathering a diverse and comprehensive dataset of skin images depicting various diseases and conditions. This dataset must be well-balanced to prevent bias in the model. Preprocessing steps such as image resizing, normalization, and augmentation are crucial to enhance the quality of the data and improve the performance of the deep learning algorithm. Additionally, techniques like noise removal, feature extraction, and data augmentation are applied to prepare the dataset for training. By meticulously curating and preprocessing the data, the machine learning model can better generalize patterns and accurately classify skin diseases, ultimately leading to more reliable diagnostic outcomes

9.2.1 STREAMLIT

Streamlit is an open-source Python library that allows you to create web applications for machine learning, data visualization, and other data-driven tasks with minimal effort. It is designed to make the process of building interactive web apps simple and efficient, particularly for data scientists and machine learning engineers who may not have extensive web development experience. Below is a detailed explanation of Streamlit along with its key features:

Simplicity and Ease of Use:

- Streamlit's primary goal is to simplify the process of building web applications, especially for those with a background in Python programming.
- Users can create interactive web apps using familiar Python scripting without needing to learn complex web development frameworks or languages like HTML, CSS, or JavaScript.

Rapid Prototyping:

- With Streamlit, you can quickly prototype ideas, visualize data, and share insights by writing a few lines of Python code.
- It facilitates an iterative development process where changes can be made and instantly viewed in the browser, enabling faster experimentation and development.

Built-in Components:

- Streamlit provides a rich set of built-in components for creating interactive user interfaces, including widgets like sliders, buttons, dropdowns, text inputs, and more.
- These components can be easily added to your app using simple Python function calls, allowing for dynamic user interactions without writing any HTML or JavaScript code.

Automatic Reactivity:

- Streamlit automatically re-executes the Python script whenever a widget's value changes, ensuring that the app's state stays in sync with user inputs.

- This reactive behavior eliminates the need for manual event handling or callbacks, making the development process more straightforward and intuitive.

Integration with Data Science Libraries:

- Streamlit seamlessly integrates with popular Python libraries for data manipulation, analysis, and machine learning such as Pandas, Matplotlib, Plotly, TensorFlow, PyTorch, and more.
- Users can leverage these libraries within their Streamlit apps to perform data processing, visualization, model training, and inference.

Customizable Layouts and Styling:

- While Streamlit emphasizes simplicity, it also offers flexibility in terms of layout and styling.
- Users can arrange components in custom layouts, adjust styling using CSS, and even embed HTML or Markdown content within their apps to create rich, multimedia experiences.

Shareability and Deployment:

- Streamlit apps can be easily shared with others by simply sharing the Python script or deploying them to various platforms such as Streamlit Sharing, Heroku, AWS, or Google Cloud Platform.
- Deployment is streamlined with built-in support for containerization and cloud deployment services, allowing users to showcase their work or deploy production-grade applications with minimal hassle.

Community and Ecosystem:

- Streamlit has a vibrant community of users and contributors who actively share tips, tutorials, and extensions.
- Users can leverage community-built components, themes, and utilities to enhance their apps and streamline development further.

9.2.2 HTML

HTML stands for HyperText Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language.

Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

Basic Construction of an HTML Page:

These tags should be placed underneath each other at the top of every HTML page that you create.

- `<!DOCTYPE html>` — This tag specifies the language you will write on the page. In this case, the language is HTML 5.
- `<html>` — This tag signals that from here on we are going to write in HTML code.
- `<head>` — This is where all the metadata for the page goes — stuff mostly meant for search engines and other computer programs.
- `<body>` — This is where the content of the page goes.
- Inside the `<head>` tag, there is one tag that is always included: `<title>`, but there are others that are just as important:
- `<title>` This is where we insert the page name as it will appear at the top of the browser window or tab.
- `<meta>` This is where information about the document is stored: character encoding, name (page context), description.

`<<head>`

`<title>My First Webpage</title>`

`<meta charset="UTF-8">`

`<meta name="description" content="This field contains information about your page. It is usually around two sentences long.">`

```
<meta name="author" content="Conor Sheils">
```

```
</header> “
```

Adding Content

Next, we will make `<body>` tag.

The HTML `<body>` is where we add the content which is designed for viewing by human eyes.

This includes text, images, tables, forms and everything else that we see on the internet each day.

How to Add HTML Headings To Your Web Page

In HTML, headings are written in the following elements:

- `<h1>`
- `<h2>`
- `<h3>`
- `<h4>`
- `<h5>`
- `<h6>`

As you might have guessed `<h1>` and `<h2>` should be used for the most important titles, while the remaining tags should be used for sub-headings and less important text. Search engine bots use this order when deciphering which information is most important on a page.

Creating Your Heading

Let's try it out. On a new line in the HTML editor, type:

```
<h1>Welcome to My Page</h1>
```

And hit save. We will save this file as **“index.html” in a new folder called “my webpage.”**

The Moment of Truth: Click the newly saved file and your first ever web page should open in your default browser. It may not be pretty it's yours... all yours. *Evil laugh*

Add Text In HTML

Adding text to our HTML page is simple using an element opened with the tag `<p>` which creates a new paragraph. We place all of our regular text inside the element

- `<p>` - Element Meaning Purpose
- `` - Bold Highlight important information
- `` - Strong Similarly to bold, to highlight key text
- `<i>` - Italic To denote text
- `` - Emphasised Text Usually used as image captions
- `<mark>` - Marked Text Highlight the background of the text
- `<small>` - Small Text To shrink the text
- `<strike>` - Striked Out Text To place a horizontal line across the text
- `<u>` - Underlined Text Used for links or text highlights
- `<ins>` - Inserted Text Displayed with an underline to show an inserted text
- `<sup>` - Superscript Text Another typographical presentation style

When we write text in HTML, we also have a number of other elements we can use to control the text or make it appear in a certain way.

Add Links In HTML

As you may have noticed, the internet is made up of lots of links Almost everything you click on while surfing the web is a link takes you to another page within the website you are visiting or to an external site. Links are included in an attribute opened by the `<a>` tag. This element is the first that we've met which uses an attribute and so it looks different to previously mentioned tags.

```
<a href="http://www.google.com">Google</a>
```

Image Tag

In today's modern digital world, images are everything. The `` tag has everything you need to display images on your site. Much like the `<a>` anchor element, `` also contains an attribute. The attribute features information for your computer regarding the source, height, width and alt text of the image

```

```

9.3 DATABASE

In the context of hosting a Streamlit application, incorporating a JSON database to store user signup information and facilitating image uploads adds essential functionality for managing user data. By integrating these features, the application becomes capable of not only capturing user details upon signup but also storing images uploaded by patients.

The JSON database serves as a structured repository for storing user information, ensuring easy retrieval and manipulation for future use. Concurrently, dedicating a specific folder within the application's directory to store uploaded images provides an organized approach to manage these assets.

This approach simplifies the process of associating uploaded images with corresponding user records, enhancing the overall functionality and usability of the Streamlit application. Additionally, by centralizing user data and uploaded images within the application's environment, it fosters a streamlined and cohesive user experience, contributing to the overall effectiveness and efficiency of the application.

```
},
{
  "name": "aa",
  "email": "aa",
  "age": 20,
  "sex": "Male",
  "password": "aa",
  "report": null,
  "precautions": null,
  "skin": null
},
{
  "name": "a4",
  "email": "a4",
  "age": 21,
  "sex": "Male",
  "password": "a4",
  "report": "Skin Disease Report:\n\nThere are indications of melanoma, the most common type of skin cancer. Early detection and evaluation are necessary for proper diagnosis and treatment.\n\nPreventative Measures:\n- Perform regular self-examinations of the skin to detect any new moles or changes in existing ones.\n- Use broad-spectrum sunscreen (SPF 30 or higher) daily, even on cloudy days.\n- Avoid tanning beds and limit direct sunlight exposure, especially during peak hours (10 AM to 4 PM).\n- Seek urgent evaluation by a dermatologist or oncologist for biopsy and follow-up appointments",
  "precautions": [
```

Fig 9.1 Data Stored in Json File

9.4 SECURITY

Ensuring the security of user data, including email addresses, is paramount when developing a Streamlit application. Implementing a mechanism to verify whether an email is already present in the system adds an additional layer of security and helps prevent duplicate or unauthorized access.

To achieve this, you can incorporate server-side validation logic within the Streamlit application. This validation process can involve checking the provided email against existing records in the JSON database before allowing a user to proceed with the signup process. If the email is already present in the database, appropriate feedback can be provided to the user, indicating that the email is already registered and prompting them to use a different email address.

Additionally, you can employ client-side validation techniques using Streamlit's input widgets to perform basic checks on the email format and ensure it meets specific criteria (e.g., correct syntax, required fields). However, it's essential to understand that client-side validation alone may not be sufficient for ensuring security, as it can be bypassed by malicious users. Therefore, server-side validation remains critical for robust security measures.

Furthermore, implementing secure authentication mechanisms such as password hashing and encryption can enhance the overall security of user data within the Streamlit application. By adhering to best practices in web security and regularly updating security measures to address potential vulnerabilities, you can maintain the integrity and confidentiality of user information within the Streamlit environment.

CHAPTER - X

CONCLUSION

CHAPTER 10

CONCLUSION

10.1 RESULT

In conclusion, the development and implementation of a deep learning-based system for skin disease detection represents a significant advancement in the field of healthcare. By harnessing the power of artificial intelligence and data analysis, this innovative approach offers a more efficient and accurate means of diagnosing and treating various skin conditions.

Through the analysis of vast amounts of medical data and images, the system can identify patterns and indicators associated with different skin diseases, enabling healthcare professionals to provide timely and targeted interventions. This not only streamlines the diagnostic process but also enhances the overall quality of care for patients.

Additionally, the integration of deep learning algorithms allows for continuous learning and improvement of the system's accuracy over time. By leveraging technology in this manner, healthcare providers can deliver more personalized and effective treatments, ultimately improving patient outcomes and quality of life. As technology continues to advance, the potential for deep learning in skin disease identification holds great promise for transforming the healthcare landscape and enhancing medical practice.

10.2 FURTHER ENHANCEMENTS

- Furthermore, as the deep learning-based system for skin disease detection evolves, its integration with telemedicine platforms could extend its reach to underserved populations, offering remote diagnosis and treatment options.
- This expansion of accessibility has the potential to bridge gaps in healthcare disparities and ensure equitable access to dermatological care for all individuals, regardless of geographical location or socioeconomic status.
- Also, a mobile application will be developed invoking this project so that a large number of people can get benefitted from this and they can lead a healthy and happy life.

BIBLIOGRAPHY

REFERENCES

- [1] Hameed, N., Shabut, A. M., Ghosh, M. K., & Hossain, M. A. (2020). Multi-class multi-level classification algorithm for skin lesions classification using machine learning techniques. *Expert Systems with Applications*, 141, 112961.
- [2] Verma, A. K., Pal, S., & Kumar, S. (2019). Classification of skin disease using ensemble data mining techniques. *Asian Pacific journal of cancer prevention: APJCP*, 20(6), 1887.
- [3] Ahmad, B., Usama, M., Huang, C. M., Hwang, K., Hossain, M. S., & Muhammad, G. (2020). Discriminative feature learning for skin disease classification using deep convolutional neural network. *IEEE Access*, 8, 39025-39033.
- [4] Balaji, V. R., Suganthi, S. T., Rajadevi, R., Kumar, V. K., Balaji, B. S., & Pandiyan, S. (2020). Skin disease detection and segmentation using dynamic graph cut algorithm and classification through Naive Bayes classifier. *Measurement*, 163, 107922.
- [5] Vijayalakshmi, M. M. (2019). Melanoma skin cancer detection using image processing and machine learning. *International Journal of Trend in Scientific Research and Development (IJTSRD)*, 3(4), 780-784.
- [6] Li, L. F., Wang, X., Hu, W. J., Xiong, N. N., Du, Y. X., & Li, B. S. (2020). Deep learning in skin disease image recognition: A review. *Ieee Access*, 8, 208264-208280.
- [7] Allugunti, V. R. (2022). A machine learning model for skin disease classification using convolution neural network. *International Journal of Computing, Programming and Database Management*, 3(1), 141-147.
- [8] Bhadula, S., Sharma, S., Juyal, P., & Kulshrestha, C. (2019). Machine learning algorithms-based skin disease detection. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 9(2), 4044-4049.
- [9] Li, H., Pan, Y., Zhao, J., & Zhang, L. (2021). Skin disease diagnosis with deep learning: A review. *Neurocomputing*, 464, 364-393.
- [10] ALenezi, N. S. A. (2019). A method of skin disease detection using image processing and machine learning. *Procedia Computer Science*, 163, 85-92.
- [11] Nawal Soliman ALKolifi ALenezi, *Procedia Computer Science* 163 (2019) 85–92A Method Of Skin Disease Detection Using Image Processing And Machine Learning.

- [12] An automated detection of Scabies skin disease Using Image Processing and CNN
Monishanker Halder*, Hussain Moh. Emrul Kabir, Afsana Mimi Rity, and Arobindo Vowmik Dept of Computer Science and Engineering, Jashore University of Science and Technology, Bangladesh
- [13] Analysis on Convolutional Neural Network Model using Skin Disease Dataset, Proceedings of the International Conference on Sustainable Computing and Smart Systems (ICSCSS 2023) IEEE Xplore Part Number: CFP23DJ3-ART; ISBN: 979-8-3503-3360-2
- [14] Shuchi Bhadula, Sachin Sharma, Piyush Juyal, Chitransh Kulshrestha Machine Learning Algorithms based Skin Disease Detection, International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075 (Online), Volume-9 Issue-2, December 2019
- [15] Classification and detection of skin diseases based on CNN-powered image segmentation, 2023 3rd International Conference on Intelligent Technologies (CONIT) Karnataka, India. June 23-25, 2023
- [16] Human Skin Diseases Detection and Classification using CNN Conference Paper · April 2023 DOI: 10.1109/ECCE57851.2023.10101636
- [17] Multi-Class Multi-Level Classification Algorithm for Skin Lesions Classification using Machine Learning Techniques Nazia Hameeda, b, *, Antesar M. Shabut, Miltu K. Ghosh, d, M. A. Hossain a School of Computer Science, University of Nottingham b Anglia Ruskin IT Research Institute (ARITI), Anglia Ruskin University, Chelmsford, UK
- [18] SKIN LESIONS CLASSIFICATION WITH DEEP CONVOLUTIONAL NEURAL NETWORK Hoang Ho Project Report CIS 5526
- [19] 2023 2nd Edition of IEEE Delhi Section Flagship Conference (DELCON) Real-time Skin Disease Prediction System using Deep Learning Approach
- [20] 2023 International Conference on Computer Science and Emerging Technologies (CSET) - Skin Cancer Detection Using CNN

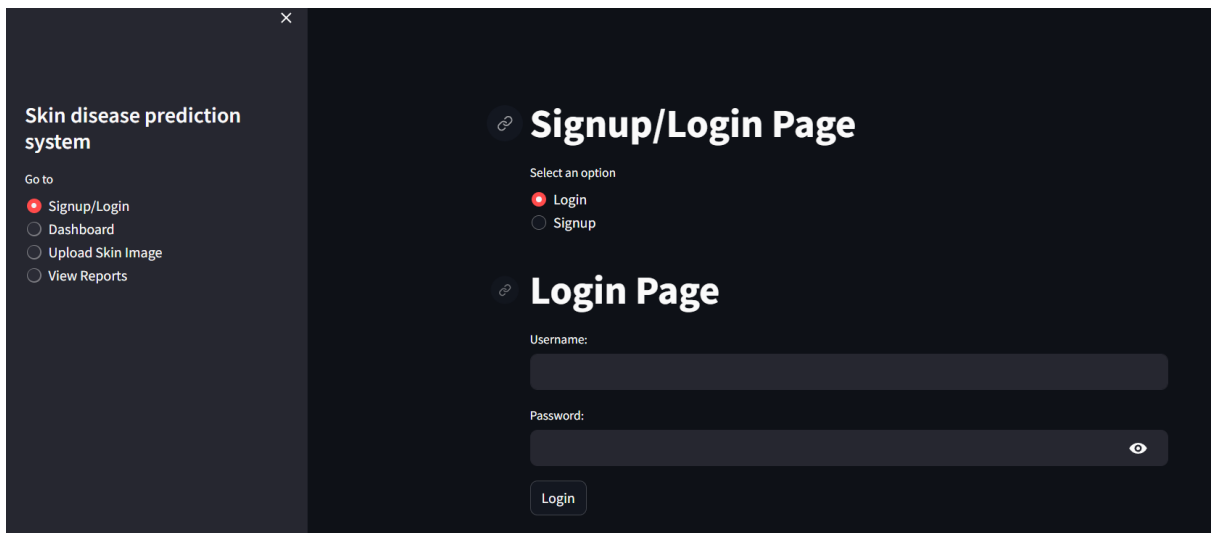
APPENDIX - I

HOST WEBSITE DEMO

Steps follows to host the website:

- 1) Open commend prompt and write the working directory to the location where the python file is stored
- 2) Run the following command <streamlit run NAME_OF_THE_PYTHON_FILE.py
- 3) A webpage will be created in your default browser where you can enter the details required

Below Screenshot shows the webpage:



The screenshot displays a web application interface for a "Skin disease prediction system". On the left, a dark sidebar contains a "Go to" menu with four options: "Signup/Login" (selected with a red dot), "Dashboard", "Upload Skin Image", and "View Reports". The main content area is divided into two sections. The top section, titled "Signup/Login Page", has a "Select an option" label and two radio buttons: "Login" (selected with a red dot) and "Signup". The bottom section, titled "Login Page", features a "Username:" label above a text input field, a "Password:" label above another text input field with a toggle eye icon, and a "Login" button at the bottom.

Webpage For Skin Disease Identification

Signup/Login Page

Select an option

☐ Login

☒ Signup

Signup Page

Fill in the details below to create an account:

Name:

Arun

Email:

arunk@gmail.com

Age:

22

- +

Sex:

☒ Male

☐ Female

☐ Other

Password:

arun123



Confirm Password:



Signup

Account created successfully! You can now login.

Signup To the Site

Signup/Login Page

Select an option

☒ Login

☐ Signup

Login Page

Username:

arunk@gmail.com

Password:

.....

Login

Login successful!

Login To the Site

Skin disease prediction system

Go to

☐ Signup/Login

☒ Dashboard

☐ Upload Skin Image

☐ View Reports

Welcome to the Dashboard, Arun!

There are 7 different classes of skin cancer which are listed below:

- Actinic keratoses
- Basal cell carcinoma
- Benign keratosis-like lesions
- Dermatofibroma
- Melanoma
- Melanocytic nevi
- Vascular lesions

User Information:

Name: Arun

Sex: Male

Age: 22

Reminder: Please upload skin images and generate a report.

Dashboard For Logged in to the Site

Skin disease prediction system

Go to

- ☐ Signup/Login
- ☐ Dashboard
- ☒ Upload Skin Image
- ☐ View Reports

There are 7 different classes of skin cancer which are listed below:

- Actinic keratoses
- Basal cell carcinoma
- Benign keratosis-like lesions
- Dermatofibroma**
- Melanoma
- Melanocytic nevi
- Vascular lesions

Upload Skin Image

Choose a skin image (JPEG/PNG)

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files

ISIC_0024327.jpg 339.2KB

×

Upload

Skin image uploaded successfully!

Melanocytic nevi

Confidence: 100.00%

Inference for the Prediction: Plot of SHAP values

SHAP value

Color scale: -0.4 (blue) to 0.4 (red)



Report For Patient in to the Site

APPENDIX - II

IMPLEMENTATION CODE

```
# Step 1: importing Essential Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from glob import glob
from PIL import Image
import tensorflow
import keras
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.models import load_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Step 2: Reading & EDA
# Set a seed for reproducibility
np.random.seed(42)
# Load the dataset
skin_df = pd.read_csv("D:\Arun\Final
Project\Dataset\HAM10000_metadata.csv")
SIZE=32
print(skin_df.head())
# null check
print(skin_df.isnull().sum())
skin_df['age'].fillna((skin_df['age'].mean()), inplace=True)
print(skin_df.isnull().sum())
#summarize
print(skin_df.info())
print(skin_df.describe(include='all'))
#eda
skin_df['dx_type'].value_counts().plot(kind='bar')

# Plotting distribution of cell types, sex, localization, and age
fig = plt.figure(figsize=(12,8))

ax1 = fig.add_subplot(221)
skin_df['dx'].value_counts().plot(kind='bar', ax=ax1)
ax1.set_ylabel('Count')
ax1.set_title('Cell Type');
```

```

ax2 = fig.add_subplot(222)
skin_df['sex'].value_counts().plot(kind='bar', ax=ax2)
ax2.set_ylabel('Count', size=15)
ax2.set_title('Sex');

ax3 = fig.add_subplot(223)
skin_df['localization'].value_counts().plot(kind='bar')
ax3.set_ylabel('Count',size=12)
ax3.set_title('Localization')

ax4 = fig.add_subplot(224)
sample_age = skin_df[pd.notnull(skin_df['age'])]
sns.histplot(sample_age['age'], kde=True, color='red', ax=ax4);
ax4.set_title('Age')

plt.tight_layout()
plt.show()

#summary function
def basic_EDA(df):
    size = df.shape
    sum_duplicates = df.duplicated().sum()
    sum_null = df.isnull().sum().sum()
    is_NaN = df.isnull()
    row_has_NaN = is_NaN.any(axis=1)
    rows_with_NaN = df[row_has_NaN]
    count_NaN_rows = rows_with_NaN.shape
    return print("Number of Samples: %d,\nNumber of Features:
%d,\nDuplicated Entries: %d,\nNull Entries: %d,\nNumber of Rows
with Null Entries: %d %.1f%%" %(size[0],size[1], sum_duplicates,
sum_null,count_NaN_rows[0],(count_NaN_rows[0]
df.shape[0])*100))

def summary_table(df):
    summary = pd.DataFrame(df.dtypes,columns=['dtypes'])
    summary = summary.reset_index()
    summary['Name'] = summary['index']
    summary = summary[['Name','dtypes']]
    summary['Missing'] = df.isnull().sum().values
    summary['Uniques'] = df.nunique().values
    return summary

def countplot(df, x, x_axis_title,y_axys_title, plot_title):
    plt.figure(figsize=(20,8))
    sns.set(style="ticks", font_scale = 1)

```

```

        ax = sns.countplot(data = df, x=x, order =
df[x].value_counts().index, palette="Blues_d")
        sns.despine(top=True, right=True, left=True, bottom=False)
        plt.xticks(rotation=0, fontsize = 12)
        ax.set_xlabel(x_axis_title, fontsize = 14, weight = 'bold')
        ax.set_ylabel(y_axis_title, fontsize = 14, weight = 'bold')
        plt.title(plot_title, fontsize = 16, weight = 'bold')

```

```

basic_EDA(skin_df)
summary_table(skin_df)

```

Data Preprocessing

Label Encoding

Labels are 7 different classes of skin cancer types from 0 to 6. We need to encode these labels to one hot vectors

1. Melanocytic nevi - nv
2. Melanoma - mel
3. Benign keratosis-like lesions -bkl
4. Basal cell carcinoma - bcc
5. Actinic keratoses - akiec
6. Vascular lesions - vasc
7. Dermatofibroma - df

Encode labels using LabelEncoder

```

le = LabelEncoder()
le.fit(skin_df['dx'])
LabelEncoder()
print(list(le.classes_))
skin_df['label'] = le.transform(skin_df["dx"])
print(skin_df.sample(10))

```

check weather the data is balanced or not

```

from sklearn.utils import resample
print(skin_df['label'].value_counts())

```

Labeling

```

df_0 = skin_df[skin_df['label'] == 0]
df_1 = skin_df[skin_df['label'] == 1]
df_2 = skin_df[skin_df['label'] == 2]
df_3 = skin_df[skin_df['label'] == 3]
df_4 = skin_df[skin_df['label'] == 4]
df_5 = skin_df[skin_df['label'] == 5]
df_6 = skin_df[skin_df['label'] == 6]

```

Upsample the minority class to balance the dataset

```

n_samples=1500

```

```

df_0_balanced = resample(df_0, replace=True, n_samples=n_samples,
random_state=42)
df_1_balanced = resample(df_1, replace=True, n_samples=n_samples,
random_state=42)
df_2_balanced = resample(df_2, replace=True, n_samples=n_samples,
random_state=42)
df_3_balanced = resample(df_3, replace=True, n_samples=n_samples,
random_state=42)
df_4_balanced = resample(df_4, replace=True, n_samples=n_samples,
random_state=42)
df_5_balanced = resample(df_5, replace=True, n_samples=n_samples,
random_state=42)
df_6_balanced = resample(df_6, replace=True, n_samples=n_samples,
random_state=42)

# Combine the balanced dataframes
skin_df_balanced = pd.concat([df_0_balanced, df_1_balanced,
                             df_2_balanced, df_3_balanced,
                             df_4_balanced, df_5_balanced,
                             df_6_balanced])
skin_df_balanced['label'].value_counts()

# Folder containing the images
image_folder = "D:\Arun\Final Project\Dataset\HAM10000_images"

# Dictionary to map image IDs to file paths
image_paths = {}
# Iterate through the files in the folder
for filename in os.listdir(image_folder):
    # Assuming the filename without extension is the image ID
    image_id = os.path.splitext(filename)[0]
    # Construct the full file path
    file_path = os.path.join(image_folder, filename)
    # Add the mapping to the dictionary
    image_paths[image_id] = file_path

# Create a dictionary to map image IDs to file paths
image_path = {os.path.splitext(os.path.basename(x))[0]: x
               for x in
glob(os.path.join('/kaggle/input/skin-cancer-mnist-ham10000',
'*.jpg'))}

# Add 'path' and 'image' columns to the balanced dataframe
skin_df_balanced['path'] =
skin_df['image_id'].map(image_paths.get)

```

```

skin_df_balanced['image'] = skin_df_balanced['path'].map(lambda
x: np.asarray(Image.open(x).resize((SIZE,SIZE))))

n_samples = 5 # number of samples for plotting
# Plotting
fig, m_axs = plt.subplots(7, n_samples, figsize = (4*n_samples,
3*7))
for n_axs, (type_name, type_rows) in zip(m_axs,
skin_df_balanced.sort_v
alues(['dx']).groupby('dx')):
    n_axs[0].set_title(type_name)
    for c_ax, (_, c_row) in zip(n_axs, type_rows.sample(n_samples,
random_state=1234).iterrows()):
        c_ax.imshow(c_row['image'])
        c_ax.axis('off')

X = np.asarray(skin_df_balanced['image'].tolist())
X = X/255. # Scale values to 0-1.
Y=skin_df_balanced['label'] #Assign label values to Y
Y_cat = to_categorical(Y, num_classes=7)

x_train, x_test, y_train, y_test = train_test_split(X, Y_cat,
test_size=0.25, random_state=32)

# ANN
# Model Building

num_classes = 7
model_cnn = Sequential()
model_cnn.add(Dense(256, input_shape=(SIZE, SIZE, 3),
activation='relu'))
model_cnn.add(Dense(128, activation='tanh'))
model_cnn.add(Dense(64, activation='relu'))
model_cnn.add(Dense(32, activation='relu'))
model_cnn.add(Dropout(0.1))
model_cnn.add(Flatten())
model_cnn.add(Dense(num_classes, activation='softmax'))
model_cnn.summary()

# Setting Optimizer and Annealer¶
model_cnn.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['acc'])

# Train the model
batch_size = 12
epochs = 25

```



```

history_cnn = model_cnn.fit(
    x_train, y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(x_test, y_test),
    verbose=2
)

loss = history_ann.history['loss']
val_loss = history_cnn.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc = history_ann.history['acc']
val_acc = history_cnn.history['val_acc']
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Prediction on test data
y_pred = model_cnn.predict(x_test)
# Convert predictions classes to one hot vectors
y_pred_classes = np.argmax(y_pred, axis = 1)
# Convert test data to one hot vectors
y_true = np.argmax(y_test, axis = 1)

n_samples_to_display = 10
# Get random indices
random_indices = np.random.choice(range(len(x_test)),
size=n_samples_to_display, replace=False)

# Plot the images
plt.figure(figsize=(15, 8))
for i, idx in enumerate(random_indices):
    plt.subplot(2, n_samples_to_display, i + 1)
    plt.imshow(x_test[idx])

```

```

        plt.title(f'Original: {np.argmax(y_test[idx])}\nPredicted:
{y_pred_classes[idx]}')
        plt.axis('off')

plt.show()

#save as dumb file
model_file_name = 'skin_model.h5'
save_dir = "C:/Users/arunk/Downloads"
os.makedirs(save_dir,exist_ok=True)
save_path = os.path.join(save_dir,model_file_name)
model_cnn.save(save_path)

```

HOST CODE (STREAMLIT CODE)

```

import numpy as np
import matplotlib.pyplot as plt
import shap
from PIL import Image
import json
import io
import os
import base64
import tensorflow
import tensorflow.keras
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from keras.preprocessing import image as img_preprocessing
from tensorflow.keras.applications.efficientnet import
preprocess_input
from tensorflow.keras.preprocessing.image import load_img,
img_to_array
import streamlit as st
from streamlit import session_state

session_state = st.session_state
if "user_index" not in st.session_state:
    st.session_state["user_index"] = 0

def signup(json_file_path="data.json"):
    st.title("Signup Page")
    with st.form("signup_form"):
        st.write("Fill in the details below to create an
account:")
        name = st.text_input("Name:")

```

```

        email = st.text_input("Email:")
        age = st.number_input("Age:", min_value=0,
max_value=120)
        sex = st.radio("Sex:", ("Male", "Female", "Other"))
        password = st.text_input("Password:", type="password")
        confirm_password = st.text_input("Confirm Password:",
type="password")

        if st.form_submit_button("Signup"):
            if password == confirm_password:
                user = create_account(name, email, age, sex,
password, json_file_path)
                session_state["logged_in"] = True
                session_state["user_info"] = user
            else:
                st.error("Passwords do not match. Please try
again.")

def check_login(username, password, json_file_path="data.json"):
    try:
        with open(json_file_path, "r") as json_file:
            data = json.load(json_file)

        for user in data["users"]:
            if user["email"] == username and user["password"] ==
password:
                session_state["logged_in"] = True
                session_state["user_info"] = user
                st.success("Login successful!")
                render_dashboard(user)
                return user

            st.error("Invalid credentials. Please try again.")
            return None
    except Exception as e:
        st.error(f"Error checking login: {e}")
        return None

def predict_skin(image_path, model):
    img = Image.open(image_path).resize((32, 32))
    img_array = img_preprocessing.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    # Make predictions using the model
    predictions = model.predict(img_array)

```

```

# Get the predicted label
predicted_label_index = np.argmax(predictions)

# Define class labels
class_labels = {
    0: "Actinic keratoses",
    1: "Basal cell carcinoma",
    2: "Benign keratosis-like lesions",
    3: "Dermatofibroma",
    4: "Melanoma",
    5: "Melanocytic nevi",
    6: "Vascular lesions"
}

predicted_label = class_labels[predicted_label_index]

return predicted_label

def load_background_batch():
    dir_path = './data'
    batch_data = []
    image_files = os.listdir(dir_path)
    # print(image_files)

    background_data = []

    for img_file in image_files:
        img_path = os.path.join(dir_path, img_file)
        # print(img_path)
        img = image.load_img(img_path, target_size=(32, 32))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        background_data.append(img_array)

    background_batch = np.vstack(background_data)

    batch_data.append(background_batch)

    return batch_data

def shap_explanation(model, img_array, background):
    explainer = shap.DeepExplainer(model, background)
    shap_values = explainer.shap_values(img_array)
    return shap_values

def show_shap(shap_values, img_array, predicted_class_idx):

```

```

if len(img_array.shape) == 3:
    img_array = np.expand_dims(img_array, axis=0)

    # Get the SHAP values for the predicted class
    shap_values_for_predicted_class =
shap_values[predicted_class_idx]

    # Plotting
    plt.figure()
    shap.image_plot(shap_values_for_predicted_class, img_array)
    plt.show()

    st.write('Inference for the Prediction: Plot of SHAP
values')
    st.pyplot(plt.gcf())

def get_predictions(uploaded,model):
    if uploaded is not None:
        global img
        img = Image.open(uploaded).resize((32,32))
        img_array = img_to_array(img)/255.0
        img_array = np.expand_dims(img_array,axis=0)
        prediction = model.predict(img_array)
        predicted_class = np.argmax(prediction)
        print(predicted_class)
        confidence = prediction[0, predicted_class]
        return predicted_class, confidence, img_array

def generate_medical_report(predicted_label):

    skin_disease_info = {
        "Actinic keratoses": {
            "report": "It appears the patient may have actinic
keratoses, which are precancerous skin lesions caused by sun
damage. Early treatment is crucial to prevent progression to
skin cancer.",
            "preventative_measures": [
                "Avoid prolonged sun exposure, especially during
peak hours",
                "Use sunscreen with a high SPF regularly, and
reapply as needed",
                "Wear protective clothing, hats, and sunglasses
when outdoors",
            ]
        }
    }

```

```

    ],
    "precautionary_measures": [
        "Schedule regular skin checks with a
dermatologist",
        "Seek medical attention if any lesions show
signs of change or growth",
    ],
},
"Basal cell carcinoma": {
    "report": "The patient may be showing signs of basal
cell carcinoma, the most common form of skin cancer. Early
detection and treatment are crucial for a good prognosis.",
    "preventative_measures": [
        "Protect skin from sun exposure with clothing
and sunscreen",
        "Avoid indoor tanning beds and booths",
        "Perform regular self-examinations of the skin
to detect any changes",
    ],
    "precautionary_measures": [
        "Consult with a dermatologist for further
evaluation and treatment options",
        "Follow recommended follow-up appointments to
monitor for recurrence or new lesions",
    ],
},
"Benign keratosis-like lesions": {
    "report": "The patient may have benign keratosis-
like lesions, which are non-cancerous growths often caused by
sun exposure. While typically harmless, monitoring for changes
is recommended.",
    "preventative_measures": [
        "Protect skin from UV radiation with sunscreen
and protective clothing",
        "Keep skin moisturized to prevent dryness and
irritation",
        "Avoid picking or scratching at lesions to
prevent infection",
    ],
    "precautionary_measures": [
        "Consult with a dermatologist for a proper
diagnosis and management plan",
        "Keep track of any changes in size, color, or
texture of the lesions for monitoring purposes",
    ],
},
"Dermatofibroma": {

```

```

        "report": "It seems the patient may have
dermatofibroma, a benign skin growth commonly found on the legs.
While typically harmless, it's important to monitor for
changes.",
        "preventative_measures": [
            "Avoid unnecessary trauma or injury to the
skin",
            "Keep skin moisturized to prevent irritation and
itching",
            "Regularly inspect the skin for any changes in
the appearance or texture of the growths",
        ],
        "precautionary_measures": [
            "Consult with a dermatologist for confirmation
and advice on management",
            "Monitor for any changes in size, color, or
texture of the dermatofibromas",
        ],
    },
    "Melanoma": {
        "report": "There are indications of melanoma, the
most serious form of skin cancer. Immediate medical attention
and further evaluation are necessary for proper diagnosis and
treatment.",
        "preventative_measures": [
            "Protect skin from UV radiation by seeking shade
and wearing sunscreen",
            "Perform regular self-examinations of the skin
to detect any new or changing moles",
            "Avoid indoor tanning beds and booths",
        ],
        "precautionary_measures": [
            "Seek urgent evaluation by a dermatologist or
oncologist for biopsy and treatment planning",
            "Follow recommended surveillance protocols for
monitoring and follow-up appointments",
        ],
    },
    "Melanocytic nevi": {
        "report": "It appears the patient may have
melanocytic nevi, commonly known as moles. While usually benign,
changes in size, shape, or color should be monitored closely.",
        "preventative_measures": [
            "Be vigilant about changes in existing moles or
the appearance of new ones",
            "Protect skin from UV radiation to prevent new
moles from forming",
        ],
    },

```

```

        "Avoid picking or scratching at moles to prevent
        irritation or infection",
    ],
    "precautionary_measures": [
        "Regularly inspect moles for any changes and
        consult with a dermatologist if concerned",
        "Consider having suspicious moles evaluated for
        biopsy or removal",
    ],
},
"Vascular lesions": {
    "report": "The patient may be presenting with
    vascular lesions, which include a variety of skin conditions
    like hemangiomas and telangiectasias. While often harmless,
    treatment may be desired for cosmetic or symptomatic reasons.",
    "preventative_measures": [
        "Avoid trauma or injury to the skin, which can
        exacerbate vascular lesions",
        "Protect skin from UV radiation with sunscreen
        and clothing",
        "Maintain good overall skin health with regular
        moisturizing and hydration",
    ],
    "precautionary_measures": [
        "Consult with a dermatologist for evaluation and
        discussion of treatment options",
        "Consider laser therapy or other interventions
        for cosmetic or symptomatic improvement",
    ],
},
}

```

```

# Retrieve medical information based on predicted label
medical_report =
skin_disease_info[predicted_label]["report"]
preventative_measures =
skin_disease_info[predicted_label]["preventative_measures"]
precautionary_measures =
skin_disease_info[predicted_label]["precautionary_measures"]

# Generate conversational medical report with each section
in a paragraphic fashion
report = (
    "Skin Disease Report:\n\n"
    + medical_report
    + "\n\nPreventative Measures:\n\n- "
    + ",\n- ".join(preventative_measures)

```



```

        + "\n\nPrecautionary Measures:\n\n- "
        + ",\n- ".join(precautionary_measures)
    )
    precautions = precautionary_measures

    return report, precautions

def initialize_database(json_file_path="data.json"):
    try:
        # Check if JSON file exists
        if not os.path.exists(json_file_path):
            # Create an empty JSON structure
            data = {"users": []}
            with open(json_file_path, "w") as json_file:
                json.dump(data, json_file)
    except Exception as e:
        print(f"Error initializing database: {e}")

def save_image(image_file, json_file_path="data.json"):
    try:
        if image_file is None:
            st.warning("No file uploaded.")
            return

        if not session_state["logged_in"] or not session_state["user_info"]:
            st.warning("Please log in before uploading images.")
            return

        # Load user data from JSON file
        with open(json_file_path, "r") as json_file:
            data = json.load(json_file)

        # Find the user's information
        for user_info in data["users"]:
            if user_info["email"] == session_state["user_info"]["email"]:
                image = Image.open(image_file)

                if image.mode == "RGBA":
                    image = image.convert("RGB")

                # Convert image bytes to Base64-encoded string
                image_bytes = io.BytesIO()

```

```

        image.save(image_bytes, format="JPEG")
        image_base64 =
base64.b64encode(image_bytes.getvalue()).decode("utf-8")

        # Update the user's information with the Base64-
        encoded image string
        user_info["skin"] = image_base64

        # Save the updated data to JSON
        with open(json_file_path, "w") as json_file:
            json.dump(data, json_file, indent=4)

        session_state["user_info"]["skin"] =
image_base64

        return

    st.error("User not found.")
except Exception as e:
    st.error(f"Error saving skin image to JSON: {e}")

def create_account(name, email, age, sex, password,
json_file_path="data.json"):
    try:
        # Check if the JSON file exists or is empty
        if not os.path.exists(json_file_path) or
os.stat(json_file_path).st_size == 0:
            data = {"users": []}
        else:
            with open(json_file_path, "r") as json_file:
                data = json.load(json_file)

        # Append new user data to the JSON structure
        user_info = {
            "name": name,
            "email": email,
            "age": age,
            "sex": sex,
            "password": password,
            "report": None,
            "precautions": None,
            "skin":None

        }
        data["users"].append(user_info)

        # Save the updated data to JSON
        with open(json_file_path, "w") as json_file:

```

```

        json.dump(data, json_file, indent=4)

        st.success("Account created successfully! You can now
login.")
        return user_info
    except json.JSONDecodeError as e:
        st.error(f"Error decoding JSON: {e}")
        return None
    except Exception as e:
        st.error(f"Error creating account: {e}")
        return None

def login(json_file_path="data.json"):
    st.title("Login Page")
    username = st.text_input("Username:")
    password = st.text_input("Password:", type="password")

    login_button = st.button("Login")

    if login_button:
        user = check_login(username, password, json_file_path)
        if user is not None:
            session_state["logged_in"] = True
            session_state["user_info"] = user
        else:
            st.error("Invalid credentials. Please try again.")

def get_user_info(email, json_file_path="data.json"):
    try:
        with open(json_file_path, "r") as json_file:
            data = json.load(json_file)
            for user in data["users"]:
                if user["email"] == email:
                    return user
        return None
    except Exception as e:
        st.error(f"Error getting user information: {e}")
        return None

def render_dashboard(user_info, json_file_path="data.json"):
    try:
        st.title(f"Welcome to the Dashboard,
{user_info['name']}!")
        st.markdown(
            """

```

```

        <div style='text-align: left;'>
            <p>There are 7 different classes of skin cancer which
are listed below:</p>
            <ul>
                <li style='color: #008000;'>Actinic keratoses</li>
                <li style='color: #ff6600;'>Basal cell
carcinoma</li>
                <li style='color: #663300;'>Benign keratosis-like
lesions</li>
                <li style='color: #ffcc00;'>Dermatofibroma</li>
                <li style='color: #6600cc;'>Melanoma</li>
                <li style='color: #990000;'>Melanocytic nevi</li>
                <li style='color: #cc0000;'>Vascular lesions</li>
            </ul>
        </div>
        """
        unsafe_allow_html=True
    )

    st.subheader("User Information:")
    st.write(f"Name: {user_info['name']}")
    st.write(f"Sex: {user_info['sex']}")
    st.write(f"Age: {user_info['age']}")

    # Open the JSON file and check for the 'skin' key
    with open(json_file_path, "r") as json_file:
        data = json.load(json_file)
        for user in data["users"]:
            if user["email"] == user_info["email"]:
                if "skin" in user and user["skin"] is not
None:
                    image_data =
base64.b64decode(user["skin"])
                    st.image(Image.open(io.BytesIO(image_dat
a))), caption="Uploaded Skin Image", use_column_width=True)

                    if isinstance(user_info["precautions"],
list):
                        st.subheader("Precautions:")
                        for precautopn in
user_info["precautions"]:
                            st.write(precautopn)

                        else:
                            st.warning("Reminder: Please upload skin
images and generate a report.")
                    except Exception as e:
                        st.error(f"Error rendering dashboard: {e}")

```

```

def fetch_precautions(user_info):
    return (
        user_info["precautions"]
        if user_info["precautions"] is not None
        else "Please upload skin images and generate a report."
    )

def main(json_file_path="data.json"):
    st.sidebar.title("Skin disease prediction system")
    page = st.sidebar.radio(
        "Go to",
        ("Signup/Login", "Dashboard", "Upload Skin Image", "View
Reports"),
        key="Skin disease prediction system",
    )

    if page == "Signup/Login":
        st.title("Signup/Login Page")
        login_or_signup = st.radio(
            "Select an option", ("Login", "Signup"),
            key="login_signup"
        )
        if login_or_signup == "Login":
            login(json_file_path)
        else:
            signup(json_file_path)

    elif page == "Dashboard":
        if session_state.get("logged_in"):
            render_dashboard(session_state["user_info"])
        else:
            st.warning("Please login/signup to view the
dashboard.")

    elif page == "Upload Skin Image":
        st.markdown(
            """
            <div style='text-align: left;'>
                <p>There are 7 different classes of skin cancer which
are listed below:</p>
                <ul>
                    <li style='color: #008000;'>Actinic keratoses</li>
                    <li style='color: #ff6600;'>Basal cell
carcinoma</li>
                    <li style='color: #663300;'>Benign keratosis-like
lesions</li>

```

```

        <li style='color: #ffcc00;'>Dermatofibroma</li>
        <li style='color: #6600cc;'>Melanoma</li>
        <li style='color: #990000;'>Melanocytic nevi</li>
        <li style='color: #cc0000;'>Vascular lesions</li>
    </ul>
</div>
"""
unsafe_allow_html=True
)
    if session_state.get("logged_in"):
        st.title("Upload Skin Image")
        uploaded_image = st.file_uploader(
            "Choose a skin image (JPEG/PNG)", type=["jpg",
"jpeg", "png"]
        )
        if st.button("Upload") and uploaded_image is not
None:
            st.image(uploaded_image, use_column_width=True)
            st.success("Skin image uploaded successfully!")
            model = load_model("skin_model.h5")
            prediction, confidence, img_array =
get_predictions(uploaded_image,model)

            classes = {
                0: "Actinic keratoses",
                1: "Basal cell carcinoma",
                2: "Benign keratosis-like lesions",
                3: "Dermatofibroma",
                4: "Melanoma",
                5: "Melanocytic nevi",
                6: "Vascular lesions"
            }

            if prediction==0:
                st.markdown(f"<h2 style='text-align: center;
color: red; font: Times New Roman;'>Actinic keratoses </h2>",
unsafe_allow_html=True)
            elif prediction==1:
                st.markdown(f"<h2 style='text-align: center;
color: red; font: Times New Roman;'>Basal cell carcinoma</h2>",
unsafe_allow_html=True)
            elif prediction==2:
                st.markdown(f"<h2 style='text-align: center;
color: red; font: Times New Roman;'>Benign keratosis-like
lesions</h2>", unsafe_allow_html=True)
            elif prediction==3:

```

```

        st.markdown(f"<h2 style='text-align: center;
color: red; font: Times New Roman;'>Dermatofibroma</h2>",
unsafe_allow_html=True)
        elif prediction==4:
            st.markdown(f"<h2 style='text-align: center;
color: red; font: Times New Roman;'>Melanoma</h2>",
unsafe_allow_html=True)
        elif prediction==5:
            st.markdown(f"<h2 style='text-align: center;
color: red; font: Times New Roman;'>Melanocytic nevi</h2>",
unsafe_allow_html=True)
        elif prediction==6:
            st.markdown(f"<h2 style='text-align: center;
color: red; font: Times New Roman;'>Vascular lesions</h2>",
unsafe_allow_html=True)

        st.markdown(
            f"<p style='text-align: center; color:
#ff6600; font-size: 1.5em;'><b>Confidence:</b>
{confidence:.2%}</p>",
            unsafe_allow_html=True)

        background_batch = load_background_batch()
        shap_values = shap_explanation(model, img_array,
background_batch)
        show_shap(shap_values, img_array, prediction)

        save_image(uploaded_image, json_file_path)

        model = load_model("skin_model.h5")
        condition = predict_skin(uploaded_image, model)
        report, precautions =
generate_medical_report(condition)

        # Read the JSON file, update user info, and
write back to the file
        with open(json_file_path, "r+") as json_file:
            data = json.load(json_file)
            user_index = next((i for i, user in
enumerate(data["users"]) if user["email"] ==
session_state["user_info"]["email"]), None)
            if user_index is not None:
                user_info = data["users"][user_index]
                user_info["report"] = report
                user_info["precautions"] = precautions
                session_state["user_info"] = user_info

```

```

        json_file.seek(0)
        json.dump(data, json_file, indent=4)
        json_file.truncate()
    else:
        st.error("User not found.")
        st.write(report)
else:
    st.warning("Please login/signup to upload a skin
image.")

elif page == "View Reports":
    if session_state.get("logged_in"):
        st.title("View Reports")
        user_info =
get_user_info(session_state["user_info"]["email"],
json_file_path)
        if user_info is not None:
            if user_info["report"] is not None:
                st.subheader("Skin Report:")
                st.write(user_info["report"])
            else:
                st.warning("No reports available.")
        else:
            st.warning("User information not found.")

```