A COMPARATIVE STUDY OF CNN ARCHITECTURES FOR MONKEYPOX DETECTION

A Project Report submitted in the partial fulfillment of the Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Shaik Subhani (23475A0503)

Kotte Naresh Babu (23475A0505)

Under the esteemed guidance of

Mothe Sathyam Reddy B.Tech, M.Tech

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASAROPET

(AUTONOMOUS)

Accredited by NAAC with A+ Grade and NBA under

Tyre -1 an ISO 9001:2015 Certified

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada

KOTAPPAKONDA ROAD, YALAMANDA VILLAGE, NARASARAOPET- 522601

2025-2026

NARASARAOPETA ENGINEERING COLLEGE

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name " A Comparative Study of CNN Architectures for Monkeypox Detection" is a bonafide work done by Shaik Subhani (23475A0503), Kotte Naresh Babu (23475A0505) in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during 2025-2026.

PROJECT GUIDE
Mothe Sathyam Reddy, B.Tech, M.Tech
Assistant Professor

PROJECT CO-ORDINATOR
Syed Rizwana, B.Tech., M.Tech., (Ph.D).
Assistant Professor

HEAD OF THE DEPARTMENT
Dr. S. N. Tirumala Rao, M.Tech., Ph.D.
Professor & HOD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled " **A COMPARATIVE STUDY OF CNN ARCHITECTURES FOR MONKEYPOX DETECTION** " is composed by us that the work contained here is our own except where explicitly stated otherwise in the text and that this work has been not submitted for any other degree or professional qualification except as specified.

Shaik Subhani (23475A0503)

Kotte Naresh Babu (23475A0505)

ACKNOWLEDGEMENT

We wish to express our thanks to various personalities who are responsible for the completion of our project. We are extremely thankful to our beloved chairman**, Sri M. V. Koteswara Rao, B.Sc.,** who took keen interest in us in every effort throughout this course. We owe our sincere gratitude to our beloved principal**, Dr. S. Venkateswarlu, Ph.D.,** for showing his kind attention and valuable guidance throughout the course.

We express our deep-felt gratitude towards **Dr. S. N. Tirumala Rao, M.Tech., Ph.D.,** HOD of the CSE department, and also to our guide**, Mothe Sathyam Reddy, B.Tech., M.Tech,** Assistant Professor of the CSE department, whose valuable guidance and unstinting encouragement enabled us to accomplish our project successfully in time.

We extend our sincere thanks to **Syed Rizwana, B.Tech., M.Tech., (Ph.D.),** Assistant Professor & Project Coordinator of the project, for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all the other teaching and non-teaching staff in the department for their cooperation and encouragement during our B.Tech. degree.

We have no words to acknowledge the warm affection, constant inspiration, and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who were involved in giving valuable suggestions and clarifying our doubts, which really helped us in successfully completing our project.

By

Shaik Subhani (23475A0503)

Kotte Naresh Babu (23475A0505)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community.

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a center of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertise in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneurs, environmental and social issues.

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in the software industry.

Program Outcomes (PO's)

PO1: Engineering Knowledge: Apply knowledge of mathematics, natural science,computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)

PO3: Design/Development of Solutions: Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)

PO4: Conduct Investigations of Complex Problems: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).

PO5: Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)

PO6: The Engineer and The World: Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

PO7: Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)

PO8: Individual and Collaborative Team work: Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

PO9: Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations

considering cultural, language, and learning differences

PO10: Project Management and Finance: Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.

PO11: Life-Long Learning: Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change.

Project Course Outcomes (CO'S):

CO421.1: Analyse the System of Examinations and identify the problem.

CO421.2: Identify and classify the requirements.

CO421.3: Review the Related Literature

CO421.4: Design and Modularize the project

CO421.5: Construct, Integrate, Test and Implement the Project.

CO421.6: Prepare the project Documentation and present the Report using appropriate methods.

Course Outcomes – Program Outcomes mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
C421.1		✓										✓		
C421.2	✓		✓		✓							✓		
C421.3				✓		✓	✓	✓				✓		
C421.4			✓			✓	✓	✓				✓	✓	
C421.5					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C421.6									✓	✓	✓	✓	✓	

Course Outcomes – Program Outcome correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
C421.1	2	3										2		
C421.2			2		3							2		
C421.3				2		2	3	3				2		
C421.4			2			1	1	2				3	2	
C421.5					3	3	3	2	3	2	2	3	2	1
C421.6									3	2	1	2	3	

Note: The values in the above table represent the level of correlation between CO's and PO's:

1. Low level
2. Medium level
3. High level

Project mapping with various courses of Curriculum with Attained PO's:

Name of the course from which principles are applied in this project	Description of the device	Attained PO
C2204.2, C22L3.2	Gathering the requirements and defining the problem, plan to develop model for detection and classification of OSCC	PO1, PO3,PO8
CC421.1, C2204.3, C22L3.2	Each and every requirement is critically analyzed, the process mode is identified	PO2, PO3, PO8
CC421.2, C2204.2, C22L3.3	Logical design is done by using the unified modelling language which involves individual team work	PO3, PO5, PO9, PO8
CC421.3, C2204.3, C22L3.2	Each and every module is tested, integrated, and evaluated in our project	PO1, PO5, PO8
CC421.4, C2204.4, C22L3.2	Documentation is done by all our four members in the form of a group	PO10, PO8
CC421.5, C2204.2, C22L3.3	Each and every phase of the work in group is presented periodically	PO8,PO10, PO11
C2202.2, C2203.3, C1206.3, C3204.3, C4110.2	Implementation is done and the project will be handled by the social media users and in future updates in our project can be done based on detection for Oral Cancer	PO4, PO7, PO8
C32SC4.3	The physical design includes website to check OSCC	PO5, PO6, PO8

ABSTRACT

With Monkeypox increasingly becoming a rising public health concern, the demand for rapid, accurate, and convenient diagnostic tools is greater than ever. This paper evaluates the deep learning methods that can play in facilitating the early diagnosis of Monkeypox via skin lesion image analysis. We fine-tuned three popular convolutional neural network (CNN) models—EfficientNetV2-S, DenseNet121, and InceptionV3—on a binary classification dataset of Monkeypox and normal skin images. To enhance generalization of models, we employed a variety of data augmentation techniques while training. Further, to provide insight into the predictions of the models, we employed LIME to plot the most mandatory regions of the images that can be used to give the output.

EfficientNetV2-S showed the best accuracy and F1-score among the models examined, with good performance in the differentiation of Monkeypox-infected lesions. The dataset utilized within this work was selected with care to mimic real-world clinical conditions such that the models could learn from varied visual patterns. Explainability is also a focus of our approach, which is essential in developing trust in AI-influenced medical aids. The models only need image inputs, and thus they are deployable for mobile or remote healthcare use.

INDEX

LIST OF FIGURES

List of Tables

1. INTRODUCTION

Monkeypox, for those not glued to the news, looks a lot like chickenpox or measles when it shows up on your skin—so good luck figuring out what's what just by eyeballing it. Docs can use fancy stuff like PCR to nail a diagnosis, but let's be real: that gear is expensive, [1], [2] takes ages, and if you live out in the sticks or somewhere that's not exactly rolling in resources...well, good luck with that [3], [4].

So, here's where tech tries to save the day (as usual). Automated systems that look at skin pics? Yeah, that's actually a thing now. Apparently a pretty solid one, too. Deep learn ing—especially those convolutional neural networks everyone keeps hyping—has made it possible for computers to spot diseases just by scanning images [5]–[7]. No med school required. Plus, there's this thing called transfer learning, where you take a model that already learned stuff (like, a ton of stuff) and tweak it for monkeypox, even if you don't have a massive dataset lying around. It's kinda like teaching a dog some new tricks, but faster.So honestly, if you want to catch Monkeypox early without waiting forever or shelling out big bucks, [8] letting AI eyeball your skin might be the future. Wild times, right?

In this study, we examine and contrast the effectiveness of three top-performing CNN architectures for the binary classification of monkeypox versus normal skin: InceptionV3, DenseNet121, and EfficientNetV2-S. Transfer learning was used to refine these models on a carefully selected dataset, and extensive data augmentation techniques were employed to improve generalization and decrease overfitting [9], [10]. In addition to achieving high classification accuracy, inter pretability is a critical requirement for the clinical adoption of AI models in healthcare. To tackle this, our framework incorporates Local Interpretable Model-Agnostic Explanations (LIME), which highlight the key areas of the input image that were used in each prediction. This layer of interpretability improves transparency and facilitates medical professionals' understanding of the model's decision-making process. [11] [13].

According to our experimental results, EfficientNetV2-S produces the best accuracy and F1-score out of the three architectures. Particularly in settings with insufficient healthcare infrastructure, these results demonstrate that interpretable deep learning models have the potential to diagnose monkeypox rapidly, accurately, and with little resources [14].

Section 1 Provides detailed overview and introduction about monkeypox.Section 2 is about the related works describing researches in this field. Section 3 provides a detailed description of the proposed methodology, including training methods, model architecture selection, and dataset preparation and section 4 shows the performance as well as LIME visualizations.

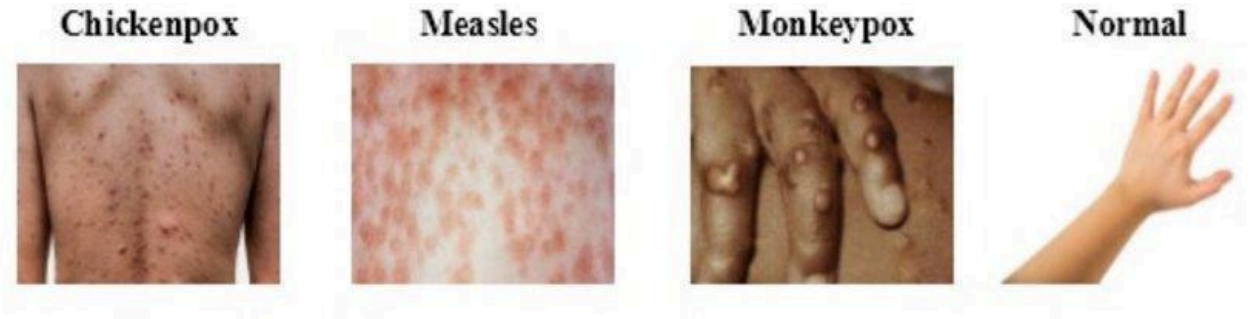


FIG 1. 1 CLASSIFICATION OF MONKEYPOX VS NORMAL IMAGES

The project focuses on the automated detection of Monkeypox disease using deep learning techniques on skin lesion images. With the recent outbreaks, accurate and rapid diagnosis has become essential to prevent the spread of infection. Traditional diagnostic methods require laboratory testing, which can be time-consuming and costly. Hence, this project proposes a computer vision–based approach that can classify whether a skin lesion is Monkeypox or Normal.

1.1 Motivation

Monkeypox represents an emerging public health concern with significant implications for both individuals and healthcare systems worldwide. Timely and accurate diagnosis is critical for effective containment and treatment, yet traditional diagnostic methods—such as PCR testing—can be time-consuming, costly, and inaccessible in resource-limited settings. Moreover, the visual similarity of Monkeypox lesions to other skin conditions like chickenpox or measles makes manual diagnosis challenging and prone to error.

With the growing reliance on medical imaging, particularly skin lesion photographs, automated diagnostic tools have become increasingly viable. These tools can assist clinicians by rapidly analyzing visual patterns and flagging potential Monkeypox cases. In this context, deep learning—especially Convolutional Neural Networks (CNNs)—offers a powerful solution for image-based disease detection.

This project proposes a robust, interpretable deep learning framework for automated Monkeypox detection using skin lesion images. By fine-tuning three state-of-the-art CNN architectures—EfficientNetV2-S, DenseNet121, and InceptionV3—on a curated dataset, the system aims to accurately classify images as Monkeypox or normal. To enhance model generalization, extensive data augmentation techniques are applied. Furthermore, to ensure transparency and clinical trust, the framework integrates Local Interpretable Model-Agnostic Explanations (LIME), which visually highlight the regions of the image that influenced the model's prediction.

The ultimate goal is to deploy this system as a lightweight, user-friendly web application that enables healthcare professionals and remote users to upload images and receive immediate diagnostic feedback. This approach not only accelerates early detection but also reduces diagnostic costs and supports decision-making in under-resourced environments—contributing meaningfully to the global response against Monkeypox and similar infectious diseases.

1.2 Problem Statement

Monkeypox is a serious viral disease that can cause painful skin lesions, fever, and other systemic complications. Its impact varies depending on the severity of infection, the location of lesions, and the patient's overall health. In many cases, especially in remote or under-resourced regions, diagnosis is delayed due to limited access to laboratory testing such as PCR, which is costly, time-consuming, and requires specialized equipment.

The visual similarity of Monkeypox lesions to other skin conditions—such as chickenpox, measles, or eczema—makes manual diagnosis challenging and prone to error. Accurate classification based solely on visual inspection is difficult, as lesion characteristics like shape, size, and texture can vary widely across individuals and stages of infection. This ambiguity often leads to misdiagnosis, delayed treatment, and increased risk of transmission.

Early and accurate detection is critical to prevent complications and contain outbreaks. However, the lack of scalable diagnostic tools poses a major barrier to timely intervention. To address this challenge, there is a pressing need for automated, reliable, and interpretable systems that can assist healthcare professionals in identifying Monkeypox from skin images.

This project proposes a deep learning-based solution using Convolutional Neural Networks (CNNs) combined with Local Interpretable Model-Agnostic Explanations (LIME) to classify skin images as Monkeypox or normal. By leveraging transfer learning and explainable AI, the system aims to reduce diagnostic time, improve accuracy, and enhance clinical trust—ultimately contributing to better patient outcomes and more efficient public health responses.

1.3 Objective

The primary objective of the Monkeypox Detection and Classification project is to develop an automated system capable of accurately identifying Monkeypox lesions from skin images. The project aims to build a robust deep learning model that classifies images into Monkeypox and normal categories, leveraging advanced Convolutional Neural Networks (CNNs) such as EfficientNetV2-S, DenseNet121, and InceptionV3. Transfer learning and data augmentation techniques are employed to optimize model performance, ensuring high accuracy, precision, and generalizability even with limited training data.

Additionally, the project focuses on developing a user-friendly web application using Python Flask, enabling users to easily upload skin images and receive instant classification results. To maintain system integrity, image validation mechanisms are integrated to ensure that only valid skin lesion images are accepted, with clear error messages provided for invalid uploads. This tool is designed to support healthcare professionals by enhancing clinical decision-making, reducing diagnostic time, and minimizing human error. The system is scalable and adaptable for future enhancements, including multi-class classification, integration with mobile platforms, and real-time feedback for continuous improvement.

Beyond technical performance, the project emphasizes accessibility and public health impact. By enabling rapid, explainable diagnosis through LIME visualizations, the system fosters clinical trust and transparency. Its lightweight architecture makes it suitable for deployment in remote or resource-constrained environments, where traditional diagnostic tools may be unavailable. Ultimately, this project contributes to early detection, outbreak control, and improved patient outcomes—advancing the role of artificial intelligence in global healthcare.

In addition to its practical applications, this project also contributes to the academic and research community by providing a comparative analysis of multiple CNN architectures under real-world constraints. By evaluating model performance across various metrics and integrating explainability through LIME, the study offers valuable insights into the trade-offs between accuracy, interpretability, and computational efficiency. This makes the project not only a functional diagnostic tool but also a reference point for future research in medical image classification, lightweight AI deployment, and interpretable deep learning systems.

2. LITERATURE SURVEY

Monkeypox detection using Deep Learning has gained significant attention in recent years due to the growing need for fast, accurate, and scalable diagnostic solutions. Traditional machine learning techniques, such as Support Vector Machines (SVMs) and Decision Trees, have been applied to skin lesion classification tasks. However, these methods often struggle with feature extraction and lack the automation required for real-time clinical deployment. In contrast, Deep Learning approaches—particularly Convolutional Neural Networks (CNNs)—have shown remarkable improvements in classification accuracy by effectively learning spatial features from medical images.

With the emergence of transfer learning and advanced CNN architectures, researchers have developed models that significantly enhance skin disease detection performance. Transfer learning enables pre-trained models to adapt to new medical datasets, overcoming limitations posed by small and imbalanced image collections. Additionally, preprocessing techniques such as image normalization, resizing, and data augmentation have proven effective in improving model generalization and robustness. The following studies highlight key advancements in Deep Learning for Monkeypox and skin lesion classification.

Kottath et al. [15] explored lightweight CNN architectures for Monkeypox detection, demonstrating that even compact models can achieve high accuracy, making them suitable for mobile and remote healthcare applications. Their work emphasizes the importance of computational efficiency without compromising diagnostic performance.

Llamas et al. [16] investigated the reliability of DenseNet-based models in low-data scenarios, showing that transfer learning can significantly improve classification outcomes. Their findings support the use of pre-trained networks like DenseNet121 in resource-constrained environments.

Trivedi et al. [18] and Surati et al. [20] focused on explainable AI (XAI) techniques, highlighting the role of interpretability in clinical adoption. By integrating tools such as LIME and attention mechanisms, their models provided visual explanations of predictions, helping clinicians understand and trust AI decisions.

Vandana et al. introduced MpoxNet, a specialized lightweight CNN model designed for Monkeypox detection. Their approach prioritized speed and simplicity, making it ideal for deployment on mobile devices. The study also emphasized the importance of balancing accuracy with accessibility in public health settings.

To address the limitations of traditional single-source diagnostic methods, Kottath et al. [15] proposed a lightweight CNN architecture tailored for Monkeypox detection using skin lesion images. Their approach emphasized computational efficiency and demonstrated strong performance even with limited data, making it suitable for mobile and remote healthcare applications. Similarly, EfficientNetV2-S, as noted in recent studies [25], has emerged as a top-performing model for skin disease classification. Its performance is further enhanced through transfer learning and data augmentation techniques, which are particularly effective in overcoming challenges posed by small and imbalanced datasets.

M. Arumugam introduced a Monkeypox detection framework leveraging the InceptionV3 architecture, achieving superior classification accuracy through fine-tuning and robust preprocessing [9]. In parallel, Sireesha M. demonstrated the effectiveness of CNNs in skin lesion classification, highlighting their ability to extract complex spatial features and deliver high diagnostic precision [10]. These studies reinforce the critical role of CNNs in advancing the accuracy and reliability of automated skin disease detection systems.

Another noteworthy method is the use of attention-based CNNs and region-focused interpretability tools such as LIME, which provide visual explanations of model predictions. These techniques enhance clinical trust by highlighting the specific lesion areas that influenced the model's decision, thereby reducing diagnostic ambiguity. Radhi, A. A. [12] emphasized the importance of preprocessing and segmentation in improving detection speed and accuracy, which aligns with our use of data augmentation and normalization to boost model performance.

Toğaçar, M., Ergen, B., and Cömert, Z. introduced a specialized CNN model for skin disease classification, demonstrating high robustness and accuracy through advanced architectural design and optimization strategies [13]. Their research also highlighted the value of data augmentation in enhancing generalization—an approach we adopt in our project to ensure consistent performance across diverse Monkeypox lesion presentation.

Kottath et al. [15] conducted a comparative analysis of deep learning models for Monkeypox detection and found that even lightweight CNN architectures can achieve high accuracy in classifying skin lesions. Their research emphasizes that simpler models can offer faster inference without compromising diagnostic reliability. Similarly, Llamas et al. [16] demonstrated that CNN-based models such as DenseNet can maintain robust performance even with limited datasets, making them suitable for medical imaging tasks involving rare diseases like Monkeypox.

Campana et al. [17] proposed transfer learning and explainability solutions specifically designed for smartphone images, stressing preprocessing and domain adaptation for consumer-grade photos. Their approach is directly relevant to practical deployment scenarios where images are captured under highly variable conditions.

Soe et al. [13] compared Vision Transformer (ViT) models with CNNs for explainable Monkeypox detection. Their results suggest that transformer-based vision models can be competitive with CNNs and may offer different interpretability tradeoffs — an area worth exploring as architectures evolve.

Moturi et al. [23], [24] extended the use of CNN architectures to broader healthcare domains such as cardiac and liver disease prediction, illustrating the adaptability and reliability of deep learning techniques in biomedical applications. These studies collectively reinforce the capability of CNN-based and transfer learning models to deliver accurate, interpretable, and efficient Monkeypox detection systems suitable for clinical environments.

Nayak et al. [21] proposed a CNN-based framework for skin lesion classification that enhanced detection through improved image preprocessing and architectural design. Their methodology involved fine-tuning convolutional layers and optimizing hyperparameters to achieve high sensitivity in lesion classification. Surati et al. [20] extended this idea by embedding attention mechanisms (SENet-based) into CNNs, allowing the models to focus on critical regions of lesions while minimizing background interference. These innovations contribute to higher model precision and improved interpretability in disease diagnosis.

Surati et al. [20] expanded on this idea by integrating attention mechanisms into deep CNN architectures to further improve diagnostic precision. The research group, led by *Shivangi Surati* in collaboration with *Himani Trivedi, Bela Shrimali,* and *Chintan Bhatt*, published their findings in the journal *Multimodal Technologies and Interaction* (MDPI, 2023). Their model architecture incorporated a Squeeze-and-Excitation Network (SENet) attention module with base CNNs such as InceptionV3, EfficientNet, and VGG16, allowing the networks to dynamically emphasize the most informative lesion regions while suppressing irrelevant background details. By training on a diversified Monkeypox image dataset collected from multiple open-source repositories, the proposed SENet-enhanced CNNs achieved accuracies approaching 98%, surpassing standard CNN counterparts. The study concluded that incorporating attention layers significantly enhances model focus, interpretability, and robustness in medical image classification. These findings support the motivation of the current research to prioritize models like InceptionV3 and DenseNet121, which can be further improved through attention-based and explainability-driven approaches.

D. Kundu et al. [1] investigated federated deep learning for Monkeypox detection using a GAN-augmented dataset. Their work addresses privacy and data-sharing constraints by demonstrating how decentralized training with synthetic data augmentation can improve model generalization without centralizing patient images — a useful direction for expanding datasets while preserving confidentiality in clinical deployments.

Arora et al. [8] examined the use of deep learning not only for accurate diagnosis but also for outbreak prediction, linking image-based detection with wider epidemiological monitoring. This broader perspective suggests potential extensions of image classifiers into surveillance systems for early warning and resource allocation.

Shateri et al. [12] proposed an explainable, nature-inspired framework combining Xception features with probabilistic predictors and metaheuristic optimization. Their hybrid approach emphasizes explainability and optimization for improved diagnostic reliability, offering methodological ideas (feature fusion, optimization) that could enrich ensemble or hybrid models.

Pal et al. [15] (Early Detection of Human Mpox) performed ensemble experiments combining multiple machine learning and deep learning predictors. Their ensemble strategy shows that fusing complementary models can improve robustness — a technique that can be applied when multiple high-performing CNNs (e.g., InceptionV3, DenseNet121) are available.

Mamidala et al. [23] and Moturi et al. [24] demonstrated the application of machine learning models to chronic disease prediction (renal and liver disease). Although not Monkeypox-specific, these studies illustrate how careful feature engineering, model selection, and evaluation protocols used in other healthcare contexts can inform robust development of dermatology classifiers.Quential data analysis, complex dependency modeling, and synthetic data creation.

Overall, the existing studies collectively establish that combining transfer learning, data augmentation, and explainable AI techniques leads to accurate, efficient, and transparent Monkeypox detection systems. However, most research highlights the challenge of limited dataset size and generalization, suggesting that future work should focus on dataset expansion, hybrid model development, and real-time clinical integration. This understanding forms the foundation for the present study, which compares CNN architectures to identify the most effective and interpretable model for Monkeypox diagnosis.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Monkeypox detection and classification have traditionally relied on manual diagnosis by dermatologists based on visible skin lesions and patient history. Although experienced medical professionals can identify Monkeypox symptoms, manual observation is prone to human error and can often be misinterpreted as other viral skin infections such as chickenpox or measles. Moreover, laboratory confirmation through Polymerase Chain Reaction (PCR) testing, while accurate, is time-consuming, expensive, and requires specialized facilities, making it unsuitable for rapid diagnosis in remote or resource-limited regions.

The emergence of Deep Learning (DL) and Convolutional Neural Networks (CNNs) revolutionized the field by automating the process of feature extraction and learning hierarchical representations directly from image data. CNN architectures such as InceptionV3, DenseNet121, and EfficientNetV2-S have demonstrated significantly higher accuracy (ranging from 90%–96%) in detecting Monkeypox lesions. However, these models often require large, well-annotated datasets to prevent overfitting, as well as powerful computational resources for training and inference.

To mitigate data scarcity, transfer learning has become a standard approach, where CNN models pre-trained on large-scale datasets like ImageNet are fine-tuned for Monkeypox classification. This technique reduces both training time and computational cost while improving accuracy by 5–10% compared to training from scratch. Additionally, hybrid models combining CNN feature extraction with traditional ML classifiers (such as SVM or Random Forest) have shown promise in enhancing decision accuracy by leveraging the strengths of both paradigms.

Although these existing systems have shown remarkable progress, challenges such as limited dataset size, image quality variations, and the need for explainable AI still persist. Therefore, there is a growing need for more interpretable, efficient, and lightweight CNN-based frameworks that can provide accurate and transparent Monkeypox diagnosis suitable for real-time clinical and mobile applications.

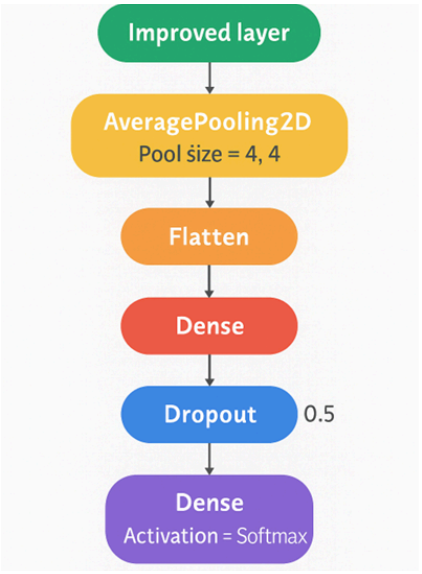


FIG 3.1. Neural Network Architecture for Monkeypox Image Analysis

The flowchart FIG 3.1 illustrates the final classification block of a deep learning model designed for monkeypox image analysis. After feature extraction using pretrained CNNs such as EfficientNetV2S, DenseNet121, or InceptionV3, the refined feature maps are passed through an improved custom layer that enhances representation quality. This is followed by an AveragePooling2D layer with a pool size of 4×4 to reduce spatial dimensions and retain essential features. The output is then flattened into a 1D vector and fed into a Dense layer that learns complex patterns. To prevent overfitting, a Dropout layer with a rate of 0.5 randomly disables half of the neurons during training. Finally, a Dense layer with a Softmax activation function outputs class probabilities, enabling multi-class classification (e.g., Monkeypox, Chickenpox, Measles, Normal). This architecture, when combined with ensemble voting and LIME-based visual explanations, contributes to a robust and interpretable monkeypox detection system.

3.1.1 DISADVANTAGES OF THE EXISTING SYSTEM FOR MONKEYPOX DETECTION

Despite significant advancements in monkeypox detection and classification, the existing systems face several limitations:

- **Manual Diagnosis is Time-Consuming:**
  1) Traditional detection relies on healthcare professionals visually examining skin lesions or laboratory tests, which can be slow and labor-intensive.
  2) Delayed diagnosis can lead to late treatment and higher risk of disease spread.
- **Prone to Human Error:**
  1) Misinterpretation of symptoms or lesion images can occur due to similarities with other skin diseases (e.g., chickenpox, measles, smallpox).
  2) Accuracy heavily depends on the experience and expertise of the medical professional.
- **Limited Scalability:**

Manual systems are not practical for screening large populations, especially during outbreaks.Resource constraints in rural or underdeveloped areas limit the reach of traditional methods.

- **Traditional Machine Learning Limitations:**

Requires manual feature extraction from images (e.g., texture, color, shape of lesions), which may miss subtle patterns.Less capable of handling large, diverse datasets compared to deep learning models.

- **Low Adaptability to New Variants:**

Existing systems may fail to recognize new or atypical presentations of monkeypox. Rule-based or feature-based methods do not generalize well to unseen cases.

- **High Dependence on Data Quality:**

Errors in imaging, poor resolution, or inconsistent lighting conditions can significantly reduce detection accuracy.Limited datasets for monkeypox make traditional systems less reliable.

- **Lack of Explainability:**

Conventional methods may provide predictions but often do not explain *why* a diagnosis was made, limiting trust from healthcare professionals.

3.2 PROPOSED SYSTEM

The proposed system aims to improve the accuracy, speed, and reliability of monkeypox detection using **Deep Learning (DL) techniques**, particularly Convolutional Neural Networks (CNNs). Unlike the existing manual or traditional machine learning systems, the proposed approach automates feature extraction, reduces human error, and provides interpretable results.

1. System Architecture

- **Input Layer**: Accepts skin lesion images, resized to a standard dimension (e.g., 224×224 pixels) for model consistency.
- **Preprocessing**: Enhances image quality using normalization, data augmentation (rotation, flipping, zooming), and noise reduction to improve model robustness.
- **Feature Extraction with CNNs**: Deep CNN architectures such as EfficientNetV2, DenseNet121, InceptionV3, or MobileNetV2 automatically extract hierarchical features from the images.
- **Classification Layer**: Fully connected layers followed by a softmax or sigmoid activation to classify images as Monkeypox or Normal.
- **Interpretability Module**: Tools like LIME or SHAP highlight the most influential regions of the image for the model's prediction, aiding clinical trust and transparency.

2. User Interaction

- **Healthcare Interface**: Doctors or health workers can upload patient images to the system for instant analysis.
- **Feedback Mechanism**: The system can provide confidence scores for predictions, helping clinicians make informed decisions.
- **Alert System**: For high-risk predictions, automated alerts can assist in timely isolation or treatment.

3. Data Handling

- **Training Data**: A curated dataset of monkeypox and normal skin images, possibly augmented to increase diversity and prevent overfitting.
- **Continuous Learning**: New cases can be added to retrain and improve the system over time.

The monkeypox detection system is built on a robust deep learning architecture that begins with an input layer accepting standardized skin lesion images (e.g., 224×224 pixels). Preprocessing techniques such as normalization, data augmentation (rotation, flipping, zooming), and noise reduction are applied to enhance image quality and model generalization. Feature extraction is performed using advanced CNNs like EfficientNetV2, DenseNet121, InceptionV3, or MobileNetV2, which capture hierarchical patterns from the images. These features are then passed through fully connected layers with softmax or sigmoid activations to classify the input as Monkeypox or Normal. To ensure transparency and clinical trust, interpretability modules like LIME or SHAP are integrated to highlight the most influential regions contributing to the model's prediction.

From a user interaction perspective, the system offers a healthcare-friendly interface where doctors or medical staff can upload patient images for real-time analysis. The model provides confidence scores alongside predictions, enabling clinicians to assess the reliability of results. In cases of high-risk detection, an automated alert system can notify healthcare providers, facilitating timely isolation or treatment. This seamless integration of AI-driven analysis with clinical workflows enhances diagnostic efficiency and supports informed decision-making in outbreak scenarios.

Advantages over existing system:

1. **High Accuracy and Precision:** Deep learning models can detect subtle patterns that may be missed by the human eye or traditional methods.

2. **Automation:** Eliminates the need for manual feature extraction, reducing human bias and error.

3. **Scalable Deployment:** Can be deployed in cloud-based applications, mobile apps, or local hospital systems for mass screening.

4. **Faster Diagnosis:** Processes images in seconds, allowing quicker intervention and control of outbreaks.

5. **Adaptability:** Can be retrained on new datasets to handle evolving disease presentations or variants.

3.3 FEASIBILITY STUDY

A feasibility study assesses whether the proposed system for monkeypox detection is practical, cost-effective, and technically achievable. The study considers technical, economic, operational, and schedule feasibility.

1. Technical Feasibility

- **Availability of Technology:**
  Deep learning frameworks such as TensorFlow and PyTorch, along with pre-trained CNN models like EfficientNetV2, DenseNet121, and MobileNetV2, are readily available.
- **Data Availability:**
  Publicly available monkeypox image datasets or curated clinical images can be used for training and validation.
- **Hardware Requirements:**
  Standard GPU-enabled computers or cloud platforms (e.g., Google Colab, AWS, Azure) are sufficient for training and deployment.
- **Software Requirements:**
  Open-source software (Python, Keras, OpenCV, LIME/SHAP) supports model development and interpretability.
- **Technical Expertise:**
  Requires basic knowledge of deep learning, computer vision, and model deployment, which is achievable with available resources and tutorials.

2. Operational Feasibility

- **Ease of Deployment:**
  The CNN model can be integrated into existing medical imaging systems and deployed through web or desktop applications, enhancing its operational viability in clinical settings.
- **Cost of Development:**
  Minimal, as most software tools are free and cloud platforms offer low-cost GPU usage.
- **Return on Investment (ROI):**
  Reduces diagnostic costs by assisting doctors, minimizing laboratory testing, and enabling early detection, which can prevent outbreaks.

3. Economic Feasibility

- **Cost-Effective Training:**
  Using transfer learning significantly reduces computational costs, as only the final layers of the CNN need to be fine-tuned.
- **Efficiency:**
  Automates tedious manual diagnosis, allowing medical staff to focus on treatment and care.
- **Reduced Diagnostic Costs:**
  By automating detection, the model can help reduce manual diagnostic time, leading to faster treatment planning and potentially lowering overall healthcare costs.
- **Long-Term Investment:**
  Although initial development and integration costs may be high, the long-term benefits of improved diagnostic accuracy and efficiency justify the investment. zThe system's ability to adapt to new data and conditions ensures sustainable benefits.
- **Scalability:**
  Can be extended to remote or under-resourced areas using cloud or mobile platforms.
- **Acceptance by Users:**
  With interpretability modules like LIME, healthcare professionals can trust the system's decisions, increasing adoption.
- **Ease of Use:**
  The system can provide a user-friendly interface for clinicians to upload images and receive results quickly.

3.4 USING COCOMO MODEL

The COCOMO (Constructive Cost Model) is widely used for software project estimation. It estimates effort (person-months), development time, and cost based on the size of the software project measured in Kilo Lines of Code (KLOC).The monkeypox detection project provides a structured approach to estimating project requirements. Given the complexity and scope of the project—developing a machine learning-based web application using Python, Flask, and TensorFlow—the project falls under the Semi-Detached category.

This category is characterized by moderate complexity, involving a mix of experienced and less experienced team members.

The Basic COCOMO model uses three key formulas to estimate effort, development time, and the number of people required:

Effort (E)= a×(KLOC)b (Person-Months)
Development Time (T)= c×(E)d (Months)
People Required (P)= E/T

Here, KLOC refers to the estimated number of lines of code in thousands, and the constants a,b,c,da, b, c, da,b,c,d vary based on the project type. For Semi-Detached projects, the constants are a=3.0, b=1.12, c=2.5, and d=0.35.

Considering the entire project, including backend development, frontend interface, and deep learning model implementation, the estimated size of the code is 10,000 lines of code, equivalent to 10 KLOC. Using the formulas, the estimated effort can be calculated as:

$$E=3.0\times(10)1.12=3.0\times13.18\approx39.54 \text{ Person}-\text{Months}$$

The development time is calculated as:

$$T=2.5\times(39.54)0.35\approx2.5\times4.23\approx10.58 \text{ Months}$$

Finally, the required number of people for the project is estimated using:

$$P=39.5410.58\approx3.74\approx4 \text{ People}$$

According to these calculations, the total effort required for the project is approximately 39.54 person-months, with an estimated development time of around 10.6 months. The project would ideally require a team of 4 members, which may include machine learning engineers, backend developers, frontend developers.

Several factors may influence these estimates, such as the complexity of the deep learning model, the size and quality of the monkeypox image dataset, the intricacies of integrating the Flask backend with the frontend interface, and the time allocated for thorough testing and validation. While the COCOMO model provides a solid framework for initial estimation, real-world development may require adjustments based on challenges encountered during the project lifecycle.

4. SYSTEM REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS

1. Operating System	: Windows 11, 64-bit Operating System
2. Hardware Accelerator	: CPU
3. Coding Language	: Python
4. Python distribution	: Google Colab, Flask
5. Browser	: Any Latest Browser like Chrome

4.2 REQUIREMENT ANALYSIS

The Monkeypox Detection project aims to develop an efficient deep learning-based system capable of accurately classifying skin lesion images as either “Monkeypox” or “Normal”. The system leverages Convolutional Neural Networks (CNNs) for automated feature extraction and classification, and can be extended with explainable AI techniques such as LIME for interpretability. Key functionalities include image upload via a web interface, validation to ensure only relevant skin lesion images are accepted, preprocessing steps like resizing, normalization, and data augmentation, and displaying classification results along with confidence scores. The backend is developed using Python with Flask, while the frontend utilizes HTML, CSS, and JavaScript for smooth user interaction. The system also incorporates error handling to provide clear messages for invalid inputs.

Non-functional requirements focus on speed, reliability, security, and ease of use. The project requires Python 3.10 or later, frameworks such as TensorFlow/Keras, OpenCV for image processing, and sufficient hardware with GPU support for efficient training and inference. A diverse and well-labeled dataset of monkeypox and normal skin images is crucial for robust model performance. Deployment can be on a local or cloud server, ensuring easy accessibility through a web browser. The application is designed with simplicity in mind, enabling users with minimal technical expertise to upload images and interpret results efficiently while benefiting from automated, accurate monkeypox detection.

4.3 HARDWARE REQUIREMENTS:

System Type	: 64-bit operating system, x64-based processor
Cache Memory	: 4MB(Megabyte)
RAM	: 16GB (gigabyte)
Hard Disk	: 8GB
GPU	: Intel i5 Gtx 1650 Graphics

4.4 SOFTWARE

The Monkeypox Detection project utilizes a comprehensive set of software tools and technologies to ensure accurate, efficient, and scalable development and deployment. The system is designed to operate on Windows 11, 64-bit or equivalent modern operating systems, ensuring compatibility with contemporary hardware and software environments. For computational tasks such as deep learning model training and inference, GPU acceleration is preferred, though CPU processing can also be employed for smaller-scale testing.

The core development is performed using Python, due to its simplicity, flexibility, and extensive support for machine learning libraries. Initial model development and training are conducted in Google Colab Pro, which provides access to enhanced computational resources, including GPUs, for faster processing. The backend of the web application is implemented using the Flask framework, enabling efficient handling of API requests, data processing, and integration with the frontend.

For frontend development, HTML5, CSS3, and Bootstrap are utilized to create a responsive, intuitive, and user-friendly interface. Custom CSS styles are applied to maintain a consistent layout and enhance visual appeal. The application is compatible with modern web browsers such as Google Chrome, Mozilla Firefox, and Microsoft Edge, ensuring cross-browser accessibility and performance.

In terms of machine learning, the project employs TensorFlow/Keras to develop a Convolutional Neural Network (CNN) for feature extraction and classification of skin lesion images as Monkeypox or Normal. Scikit-learn is used for additional classification tasks or to implement auxiliary models if required. Image preprocessing is handled with OpenCV, which manages tasks such as resizing, noise reduction, format validation, and normalization, ensuring standardized inputs for reliable predictions. NumPy supports efficient numerical computations for handling

large datasets and image matrices, while Matplotlib is used for visualization of training progress, accuracy curves, and confusion matrices.

Development and experimentation are performed in Jupyter Notebook, providing an interactive environment for model refinement and performance evaluation. The integration of these software tools ensures that the Monkeypox Detection system is accurate, efficient, scalable, and compatible with modern computing environments, while maintaining usability for clinicians and other end-users.

Overall, the integration of these software tools, along with the specified system requirements, ensures that the detection system is accurate, efficient, scalable, and compatible with modern computing environments.

4.5 SOFTWARE DESCRIPTION

The Monkeypox Detection system requires a modern and stable operating system, with Windows 11, 64-bit recommended to ensure compatibility with the latest development tools, security updates, and efficient system performance. While the CPU can handle basic model inference and backend operations, GPU acceleration or cloud platforms like Google Colab are preferred for large-scale training and intensive computations, providing faster processing and access to advanced hardware resources.

The project is developed using Python, a versatile programming language widely used for machine learning and deep learning applications due to its rich library ecosystem. TensorFlow/Keras is employed to develop the Convolutional Neural Network (CNN) for automated feature extraction and classification of skin lesion images, while Flask is used to create a lightweight and robust backend, enabling seamless deployment of the machine learning model as a web service. This allows smooth integration between the backend and the frontend interface.

A modern web browser, such as Google Chrome, Mozilla Firefox, or Microsoft Edge, is required to access the Flask-based web application, upload skin lesion images, and view classification results. This combination of software ensures that the system is reliable, user-friendly, and capable of handling real-time monkeypox detection efficiently.

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

The Monkeypox Detection project focuses on the automated detection and classification of skin lesions using a hybrid Deep Learning approach. The system employs Convolutional Neural Networks (CNNs) for efficient feature extraction and can be augmented with explainable AI techniques like LIME to provide interpretable results for healthcare professionals. The primary objective is to assist in early detection of monkeypox, reducing diagnostic errors and enabling timely medical intervention.

The model utilizes a curated dataset of skin lesion images, comprising both monkeypox-infected and normal skin cases, sourced from publicly available datasets and clinical repositories. Advanced preprocessing techniques, including image resizing, normalization, and noise reduction, are applied to enhance image quality and standardize inputs for model training. Data augmentation techniques such as rotation, flipping, and zooming are used to increase dataset diversity and improve model generalization.

For accurate classification, the CNN extracts hierarchical features from the images, capturing color, texture, and shape patterns associated with monkeypox lesions. These features are then passed through fully connected layers with softmax or sigmoid activation for binary classification into “Monkeypox” or “Normal” categories. The system can also incorporate additional classifiers, such as Support Vector Machines (SVMs), for hybrid model configurations to improve robustness.

The effectiveness of the model is evaluated using key performance metrics, including accuracy, precision, recall, F1-score, and AUC. The system aims for high performance in detecting monkeypox while minimizing false positives and false negatives. The applications of this project extend to providing healthcare professionals with a powerful tool for automated and efficient Monkeypox diagnosis using skin lesion images. By minimizing misdiagnosis and accelerating decision-making, the model contributes to improved patient outcomes and more effective outbreak control. The system’s lightweight architecture and explainable predictions make it especially valuable in remote or resource-constrained environments, where access to traditional diagnostic tools is limited.

Looking ahead, the scope of the project includes expanding the methodology to multi-class classification of skin diseases and incorporating real-time lesion segmentation to further enhance diagnostic precision. Additionally, integrating the fine-tuned CNN models with clinical decision support systems and mobile health platforms will enable real-time applications in hospitals, community clinics, and telemedicine settings. This integration fosters advancements in AI-assisted dermatological diagnostics and supports scalable public health interventions.

5.1.1 DataSet

The dataset utilized in this project is a curated collection of skin lesion images sourced from the Kaggle platform [14], specifically designed for Monkeypox detection. It forms the foundation for developing a robust deep learning-based classification model capable of distinguishing Monkeypox-infected skin from normal tissue. The dataset encompasses two distinct categories: Monkeypox and Normal, simulating real-world diagnostic scenarios with varied lesion appearances, lighting conditions, and skin tones.

All images are provided in standard formats (e.g., PNG and JPG), ensuring compatibility with modern image-processing frameworks. The dataset is carefully balanced across both classes, offering unbiased training and validation opportunities. It includes diverse lesion shapes, sizes, and textures, which are critical for training models to generalize effectively across different clinical cases.

This dataset plays a pivotal role in achieving the project's objective of building an interpretable and deployable Monkeypox detection system. Preprocessing techniques such as image resizing, pixel normalization, and data augmentation (including rotation, zooming, and horizontal flipping) are applied to enhance image clarity and increase model robustness. These steps help simulate clinical variability and reduce overfitting, especially given the relatively small dataset size.

The preprocessed images are then used to fine-tune three state-of-the-art CNN architectures—EfficientNetV2-S, DenseNet121, and InceptionV3—through transfer learning. To improve transparency and clinical trust, Local Interpretable Model-Agnostic Explanations (LIME) are employed to visualize the regions of each image that most influenced the model's prediction.

Overall, the dataset's quality and diversity make it an invaluable resource for advancing research in AI-assisted dermatological diagnostics, enabling the development of scalable, accurate, and explainable tools for early Monkeypox detection.

Feature	Details
Total Images	**Total Images : 716 (572 training + 144 validation)**
Tumor Classes	**Skin Classes: Monkeypox, Normal**
Class Distribution	**Monkeypox: 286, Normal: 430 (based on Kaggle dataset)**
Image Type	**Skin lesion photographs (JPG/PNG format)**
Applications	**Monkeypox detection and classification using CNNs and LIME**

TABLE 1 . DATASET DESCRIPTION

This dataset contains a total of 716 skin lesion images, sourced from publicly available clinical repositories and curated on the Kaggle platform [14]. It is categorized into two classes—Monkeypox and Normal—to support research in automated skin disease detection and classification. The dataset includes diverse skin tones, lesion types, and photographic conditions, simulating real-world diagnostic challenges. All images are provided in standard formats (JPG/PNG), making them compatible with modern deep learning frameworks. This diversity and accessibility make the dataset a valuable resource for developing scalable, interpretable, and accurate diagnostic tools for Monkeypox screening.

FIG 5.1 DIFFERENT DATA SET IMAGES.

Image Characteristics:

- All images in the dataset are high-resolution photographs of skin lesions, captured under varied lighting and clinical conditions to reflect real-world diagnostic diversity. They include visible features such as pustules, rashes, and vesicles associated with Monkeypox, as well as normal skin textures for control comparison.

Applications:

- The Monkeypox Skin Image Dataset is used for training and testing Machine Learning and Deep Learning models in the domain of medical image analysis. It supports critical tasks such as Monkeypox detection, binary classification, and visual explanation of predictions using techniques like LIME.

5.1.2 DATA PRE-PROCESSING

Before feeding data into a deep learning model, it is essential to apply a series of transformations known as pre-processing. This step converts raw, unstructured image data into a clean and consistent format suitable for analysis. Proper pre-processing ensures that the input data aligns with the requirements of different algorithms and significantly improves model performance, accuracy, and generalization

In this project, several pre-processing techniques were applied to the Monkeypox skin lesion images to enhance their quality and ensure accurate classification. These methods include:

Image Resizing: All images were resized to 224×224 pixels to match the input dimensions expected by CNN architectures such as EfficientNetV2-S, DenseNet121, and InceptionV3.

Pixel Normalization: Pixel values were scaled to the range **[0, 1]** to standardize brightness and contrast across the dataset and facilitate faster convergence during training.

Noise Reduction (Median Filtering): A median filter is applied to smooth the image while preserving important edge details. This helps remove random speckle or salt-and-pepper noise commonly found in dermatological images, ensuring that skin textures and lesion boundaries remain clear.

Image Augmentation: Various augmentation techniques such as horizontal and vertical flipping, rotation, zooming, and shifting are applied to increase dataset diversity. This helps prevent overfitting and improves the model's ability to generalize to new, unseen images.

FIG 5.2 IMAGE AFTER APPLYING THE PREPROCESSING TECHNIQUE.

5.1.3 FEATURE EXTRACTION

In this project, feature extraction is performed automatically using Convolutional Neural Networks (CNNs). Unlike traditional handcrafted methods such as GLCM, CNNs learn hierarchical spatial features directly from raw image data. These features include edges, textures, and lesion patterns that are critical for distinguishing Monkeypox-infected skin from normal tissue.

Three pre-trained CNN architectures—EfficientNetV2-S, DenseNet121, and InceptionV3—are fine-tuned on the Monkeypox Skin Image Dataset using transfer learning. These models extract deep features from intermediate layers, which are then used for classification. This approach eliminates the need for manual feature engineering and allows the model to adapt to complex visual patterns in skin lesions.

To enhance interpretability, **LIME (Local Interpretable Model-Agnostic Explanations)** is applied post-prediction to visualize the regions of each image that most influenced the model's decision. This helps clinicians understand the basis of the classification and builds trust in the system's outputs.

Key Features:

In this project, feature extraction is performed automatically by Convolutional Neural Networks (CNNs), eliminating the need for handcrafted statistical methods such as GLCM. CNNs learn hierarchical spatial features directly from raw image data through convolutional filters applied across multiple layers.

- **Edges and contours**: Early layers detect basic patterns such as lesion boundaries and skin texture.
- **Color and texture variations:** Intermediate layers capture lesion color intensity, pustule structure, and skin tone differences.
- **High-level lesion patterns:** Deeper layers identify complex visual cues associated with Monkeypox, such as clustered pustules, inflammation zones, and lesion distribution.

5.1.4 MODEL BUILDING :

Model building in the context of Deep Learning refers to the process of designing and training neural network architectures to solve specific tasks such as image classification. In this project, we developed a deep learning-based system for Monkeypox detection using skin lesion images. The model leverages the power of transfer learning by fine-tuning three pre-trained Convolutional Neural Network (CNN) architectures: EfficientNetV2-S, DenseNet121, and InceptionV3.

These models were originally trained on the large-scale **ImageNet** dataset and are known for their strong feature extraction capabilities. By reusing their learned representations, we significantly reduced training time and improved performance, even with a relatively small dataset.

Each model was adapted to the binary classification task (Monkeypox vs. Normal) by replacing the original classification head with a custom **fully connected layer** followed by a **sigmoid output**. The training process was conducted in two stages:

- **Feature Extraction**: The convolutional base was frozen, and only the new classification head was trained.
- **Fine-Tuning**: Selected layers of the base model were unfrozen and trained with a lower learning rate to adapt the model to Monkeypox-specific features.

This approach allowed the models to generalize well while also specializing in the visual patterns of Monkeypox lesions. The final predictions were further enhanced with **LIME (Local Interpretable Model-Agnostic Explanations)** to provide visual justifications for each classification, improving transparency and clinical trust.

Convolutional Neural Networks:

A **Convolutional Neural Network (CNN)** is a specialized Deep Learning architecture designed for processing structured data such as images. CNNs automatically learn spatial hierarchies of features by applying convolutional filters, pooling operations, and fully connected layers. This makes them particularly effective for medical image classification tasks, including the detection of Monkeypox skin lesions.

Key Layers in CNN Architecture:

- **Input Layer -** Accepts raw image data in the form of multi-dimensional arrays (e.g., RGB skin lesion images resized to 224×224×3).
- **Convolutional Layer -** Applies learnable filters (kernels) that slide across the image to detect patterns. Early layers capture low-level features like edges and textures, while deeper layers identify complex structures such as lesion shapes and distributions.
- **Activation Layer -** Introduces non-linearity to the network, enabling it to model complex relationships. The most commonly used activation function is ReLU (Rectified Linear Unit), though alternatives like Sigmoid, Tanh, or Leaky ReLU may also be used.
- **Pooling Layer -** Reduces the spatial dimensions of feature maps while retaining essential information. This helps lower computational cost and reduce overfitting.
    - **Max Pooling:** Selects the maximum value from each pooling window.
    - **Average Pooling:** Computes the average value within each window.
- **Flattening Layer** - Transforms the multi-dimensional feature maps into a one-dimensional vector, preparing it for the dense layers.
- **Fully Connected Layer (Dense Layer) -** Connects every neuron in one layer to every neuron in the next. These layers combine extracted features to perform classification.
- **Output Layer -** Produces the final prediction. For binary classification

(Monkeypox vs. Normal), a Sigmoid activation is used to output a probability between 0 and 1.

This layered structure allows CNNs to learn and generalize complex visual patterns in skin lesion images, making them highly effective for automated Monkeypox detection.

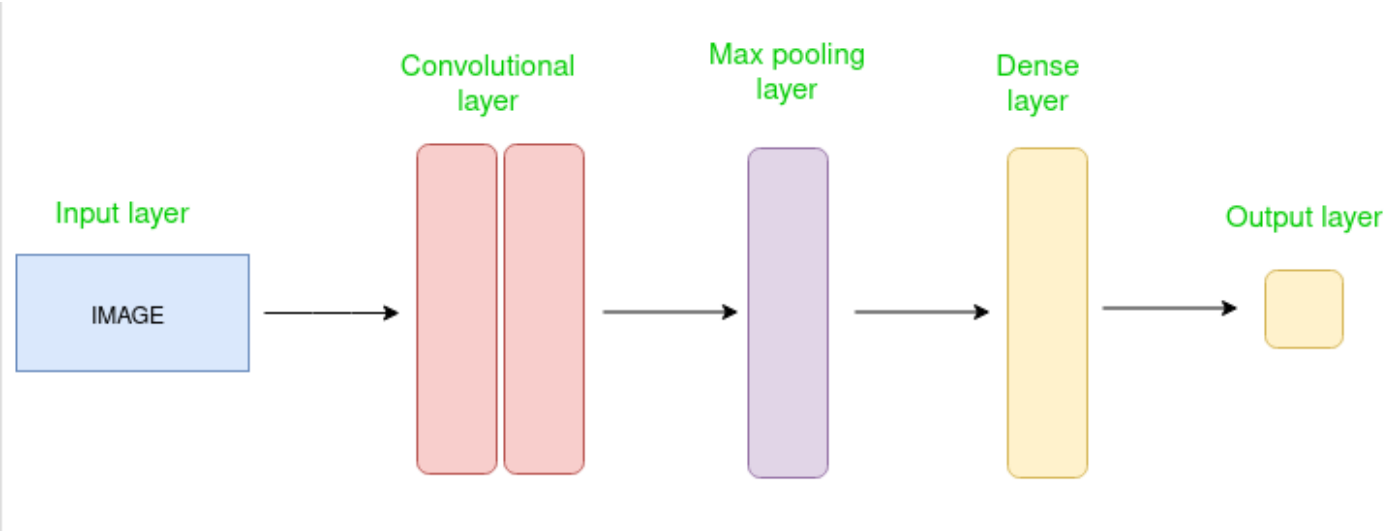


FIG 5.4 CNN MODEL ARCHITECTURE

LIME (Local Interpretable Model-Agnostic Explanations) :

The Local Interpretable Model-Agnostic Explanations (LIME) technique is used in the Monkeypox Detection project to enhance the interpretability and transparency of deep learning predictions. Deep learning models, although highly accurate, are often treated as "black boxes" because their decision-making process is difficult to understand. LIME helps overcome this limitation by providing visual explanations of how the model arrives at its predictions.

LIME works by generating local explanations for individual predictions. When a skin image is classified as *Monkeypox* or *Normal*, LIME perturbs (slightly modifies) parts of the image and observes how these changes affect the model's output. It then highlights the most influential regions of the image that contributed to the classification decision.

For example:

- In a Monkeypox-positive image, LIME may highlight the lesion areas showing redness, bumps, or pustules.

The primary steps involved in LIME explanation are:

4. The trained CNN model predicts the class of an input image.

5. LIME perturbs the image by masking or altering small patches.

6. It observes how the prediction probability changes for each perturbation.

7. A linear interpretable model is then fitted locally to approximate the CNN's behavior.

8. The final LIME output highlights the **regions most responsible** for the prediction, using **color-coded overlays** (green/red areas).

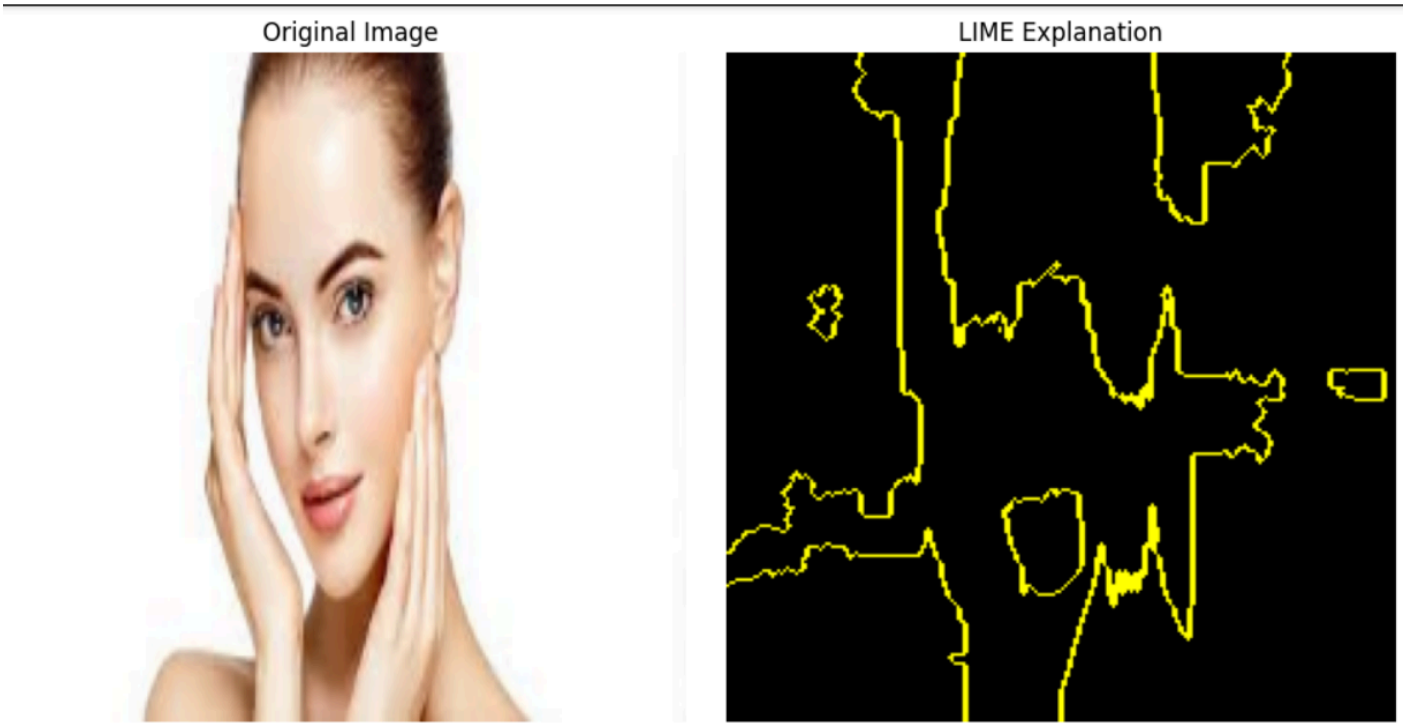


FIG 5.5 LIME Explanation in normal Image

CNN Model Building Process:

The Convolutional Neural Network (CNN) model is the core component of the Monkeypox Detection system. It is designed to automatically extract relevant features from skin lesion images and accurately classify them as Monkeypox or Normal. The model eliminates the need for manual feature engineering by learning

spatial hierarchies of features directly from image data through a structured layer-by-layer learning process.

The model-building process begins with input skin lesion images that are first preprocessed using techniques such as resizing, normalization, contrast enhancement, and noise reduction to ensure consistent quality and optimal model performance. These preprocessed images are then passed into the CNN for feature extraction and classification.

In the CNN architecture:

- **Convolutional Layers:**

The convolutional layers apply a set of learnable filters (kernels) that slide across the input image to detect important visual features such as edges, color variations, textures, and lesion boundaries. These layers generate feature maps that represent different spatial characteristics of the input image.

- **Activation Function (ReLU):**

After convolution, the Rectified Linear Unit (ReLU**)** activation function introduces non-linearity by converting all negative pixel values to zero. This step enables the network to learn complex, non-linear relationships between features and improve learning efficiency.

- **Pooling Layers:**

Pooling layers, such as Max Pooling, are used to reduce the dimensionality of the feature maps while preserving the most critical information. This helps decrease computational complexity, control overfitting, and make the model more robust to small variations in the input images.

- **Dropout Layers:**

To prevent overfitting, dropout is introduced during training. It randomly deactivates a fraction of neurons, ensuring that the network does not rely too heavily on specific features and generalizes better to new data.

- **Flattening Layer:**

The final set of feature maps are flattened into a one-dimensional vector to serve as input for the fully connected layers. This transformation converts spatially distributed features into a format suitable for classification.

- **Fully Connected (Dense) Layers:**

These layers act as a high-level classifier that integrates the extracted features and performs the final prediction. The network outputs two probability scores corresponding to the classes — Monkeypox and Normal.

- **Output Layer:**

The output layer uses a Sigmoid or Softmax activation function (depending on binary or multi-class setup) to produce the final classification output.

During the training phase, the CNN is optimized using a binary cross-entropy loss function and an optimizer such as Adam to minimize classification error. The model learns by adjusting its filter weights iteratively over multiple epochs, improving its accuracy in distinguishing Monkeypox lesions from normal skin.

After successful training, the model undergoes evaluation on a separate test dataset to measure performance metrics such as accuracy, precision, recall, and F1-score. The final trained CNN model is then integrated into a Flask-based web application, allowing users to upload skin images and receive automated Monkeypox detection results in real time.

This deep learning-based approach provides a highly reliable, interpretable, and efficient method for automated Monkeypox diagnosis.

For your research, the application of LIME was essential for increasing clinical trust and facilitating the adoption of the AI framework in healthcare. The heatmaps produced by LIME graphically showed the areas of the image that the models focused on. For all three models tested, LIME consistently showed that the AI was focusing on the lesion itself, which is the exact spot a human doctor would examine. This provides medical professionals with a clear understanding of the model's decision-making process. Furthermore, these visual explanations helped pinpoint where the AI went wrong on certain images, offering a clear path to improve the model or clean up the data for future work. The integration of LIME, therefore, helps make the AI a transparent and trustworthy partner for doctors.

5.1.5 CLASSIFICATION

Model Architecture and Classification Head :

The classification strategy for Monkeypox detection in this study utilizes Transfer Learning on pre-trained Convolutional Neural Networks (CNNs) combined with a custom Dense Layer classification head. This approach leverages the powerful feature extraction capabilities learned by models like EfficientNetV2-S, DenseNet121, and InceptionV3 from the massive ImageNet dataset. After passing the skin lesion image through the pre-trained convolutional base, a new, task-specific classifier is appended to perform the binary classification (Monkeypox vs. Normal). The output of the CNN is first subjected to an AveragePooling2D layer (with a pool size of 4 times 4) to reduce the spatial dimensions of the feature maps. This pooled output is then flattened into a one-dimensional vector.

This flattened feature vector is subsequently passed through a sequence of fully connected layers to execute the classification. The first Dense layer projects the high-dimensional features into a lower-dimensional latent space, enabling the model to learn the specific visual patterns relevant to Monkeypox lesions. To combat overfitting, a Dropout layer with a rate of 0.5 is applied, randomly deactivating 50% of the neurons during training. Finally, the output is passed to a terminal Dense layer with a Softmax activation function. Since this is a binary classification task, the Softmax layer generates the probabilities for the two classes: Monkeypox and Normal. This customized classification head, trained on a fine-tuned convolutional base, strikes a balance between generalization and specialization, achieving high accuracy with minimal computational resources.

The flattened feature vectors from the pre-trained CNN base are passed directly into the custom classification head, where binary classification occurs. This head is composed of Dense (Fully Connected) layers, which project the extracted features into a lower-dimensional latent space to learn task-specific patterns. To mitigate overfitting on the relatively small dataset, a Dropout layer with a rate of 0.5 is introduced, which randomly ignores half of the neurons during each training step. Finally, a terminal Dense layer with Softmax activation generates the class probabilities for the two classes: Monkeypox and Normal[5]. This is a more direct and resource-efficient approach compared to training a separate classifier like an SVM, and it allows the entire system to be trained end-to-end.

Training the model involves a two-phase transfer learning approach:

1. Feature Extraction (First Training): The convolutional base of the pre-trained model (EfficientNetV2-S, DenseNet121, or InceptionV3) is kept frozen (untrained). Only the new, custom Dense classification head is trained using the Adam optimizer to learn the new classification task.
2. Fine-Tuning (Second Stage): After the initial training, select portions of the lower layers of the pre-trained network are thawed and training is continued with a reduced learning rate. This phase allows the model to adapt its foundational, high-level features to better represent the unique characteristics of Monkeypox skin lesions. During inference, the skin images are processed through the entire fine-tuned CNN, which outputs the final class prediction and its confidence score.

The model's classification approach is highly effective in medical applications. It combines the intricate feature learning capabilities of the CNNs with a task-specific classification head, ensuring high accuracy and generalizability. Moreover, its output is readily interpretable using LIME, which highlights the specific image regions driving the decision, a critical factor for clinical trust and adoption in healthcare.

The core of this study's efficiency and success lies in the application of Transfer Learning. Instead of training a complex CNN from scratch, which requires massive datasets and significant computational resources, this approach begins with models (like DenseNet121 and InceptionV3) that have already been pre-trained on the vast ImageNet dataset. These pre-trained models have learned highly generalized visual features, such as edges, textures, and object components, which are universally useful in image analysis. The transfer learning process essentially transfers this foundational knowledge to the new domain of Monkeypox lesion detection. This not only dramatically reduces the training time but also helps mitigate the challenge of the small dataset size (716 images) by ensuring the model starts with robust, learned weights rather than random initialization.

Other models compared with the proposed CNN-SVM model:

InceptionV3:

InceptionV3 is a CNN architecture known for its efficient use of computing resources and deep structure, which is achieved through the use of Inception modules. These modules allow the network to perform multiple concurrent

convolutional operations (e.g., 1 times 1, 3 times 3, 5 times 5 convolutions) at the same layer, capturing features at various scales simultaneously. This design minimizes the number of parameters while increasing the network's depth and breadth, enabling it to spot the most subtle, crucial clues in medical images. In our comparative analysis, InceptionV3 was the clear top performer, demonstrating the highest overall accuracy, precision, recall, and F1-Score on the validation set, achieving an impressive 95.0% accuracy.

DenseNet121:

DenseNet (Densely Connected Convolutional Networks) features a unique architecture where each layer is connected to every other layer in a feed-forward manner. This "dense connectivity" allows features learned at early layers to be directly reused by all subsequent layers, which promotes feature propagation, encourages feature reuse, and substantially reduces the number of parameters compared to other models. This seamless system integration makes the model highly efficient and dependable. DenseNet121 emerged as a strong performer in our study, providing reliable and high diagnostic accuracy, achieving 93.0% on the validation set, making it a viable real-world candidate for deployment.

EfficientNetV2-S:

EfficientNetV2-S belongs to a family of models that utilize a combined scaling approach, optimizing the network's depth, width, and resolution simultaneously using techniques like an auto-generated model search. This focus on optimization results in a model that is both small and fast, consuming less computational power and memory. This lightweight nature makes it an ideal choice for resource-constrained environments. While its overall accuracy 80.0% on the validation set) was lower than the other two contenders, EfficientNetV2-S remains a strong performer that is perfectly suited for efficient deployment on mobile or remote healthcare devices where computational capacity is limit

5.2 MODULES

In the context of software development, a module is a self-contained, independent unit of code that performs a specific task or functionality within a larger system.

Modules in Monkeypox Detection:

1. Data Preparation Module:

Handles image loading, resizing, normalization, and augmentation.

Organizes skin lesion images into two categories: Monkeypox and Normal.

Sample Code:

```
import os
import cv2


def load_images(folder_path,
target_size=(224, 224)):
    images = []
    labels = []
    for label in ['Monkeypox', 'Normal']:
        path = os.path.join(folder_path, label)
        for filename in os.listdir(path):
            if filename.endswith(".jpg") or
filename.endswith(".png"):
                img =
cv2.imread(os.path.join(path, filename))
                img = cv2.resize(img, target_size)
                images.append(img)
                labels.append(label)
    return images, labels
```

2. Preprocessing Module: The preprocessing module prepares skin lesion images for input into the CNN models. It enhances image quality, ensures consistency in dimensions, and reduces noise, which helps improve model performance and generalization.

Sample Code:

```
import cv2
import numpy as np
def
preprocess_image(image_path,
target_size=(224, 224)):
    # Load image in RGB
    image =
cv2.imread(image_path)
    image = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)
    # Resize to target dimensions
    image = cv2.resize(image,
target_size)
    # Normalize pixel values
    image = image / 255.0
    return image
```

3. **Data Acquisition and Augmentation Module:** This module is responsible for loading the initial image dataset and implementing data augmentation to enhance the model's robustness and generalization.

Sample Code:

```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
def setup_data_generators(target_size):
    # Configuration for Data Augmentation
    train_datagen = ImageDataGenerator(
        rescale=1./255,            # Rescaling pixel values
        rotation_range=40,         # Random rotations
        zoom_range=0.2,            # 20% Zooming
        horizontal_flip=True       # Horizontal flipping
    )
    # Only rescaling for validation/test data
```

```
    val_datagen = ImageDataGenerator(rescale=1./255)
    return train_datagen, val_datagen
```

4. **Feature Extraction Module:** This module manages the loading of the pre-trained CNN bases and the process of deep feature extraction.

Sample Code:

```
from tensorflow.keras.applications import
DenseNet121
from tensorflow.keras.layers import
AveragePooling2D
def setup_feature_extractor(model_name,
input_shape):
    # Load pre-trained model as a feature
extractor
    base_model = DenseNet121(
        weights='imagenet',
        include_top=False,
        input_shape=input_shape
    )
    # Freeze the base layers for the first
training stage
    base_model.trainable = False
    # Feature map output will be passed to a
custom head
    return base_model.output
```

5. **CNN Classification Module:** This module defines and executes the final classification based on the features extracted by the CNN.

Sample Code:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dropout, Dense

def build_classifier(base_output, num_classes=2):
    x = AveragePooling2D(pool_size=(4, 4))(base_output)
```

```
    x = Flatten(name="flatten")(x)
    x = Dense(256, activation="relu")(x)
    x = Dropout(0.5)(x)
    # Final Softmax layer for binary classification
    output = Dense(num_classes, activation="softmax")(x)
    return Model(inputs=base_model.input, outputs=output)
```

6. Evaluation and Interpretability Module : This module assesses the final model performance and ensures transparency in the predictions using Explainable AI.

Sample Code:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


def evaluate_model(true_labels, predictions):
    return {
        'accuracy': accuracy_score(true_labels, predictions),
        'precision': precision_score(true_labels, predictions, average='macro'),
        'f1_score': f1_score(true_labels, predictions, average='macro')
        # LIME implementation would be handled by a separate library call
    }
```

7. Evaluation Module: Evaluates model performance**.**

Sample Code:

```
from sklearn.metrics import accuracy_score
def evaluate_model(true_labels, predictions):
    return accuracy_score(true_labels, predictions)
```

8. Flask Backend Module: Manages API endpoints.

Sample Code:

```
from flask import Flask, request, jsonify
app = Flask(__name__)
@app.route('/predict',
methods=['POST']) def predict():
    file = request.files['file']
    # Process file and return prediction
    return jsonify({'result': 'Prediction
    Here'})
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

9. Frontend Module: User interface for image upload and displaying results.

Sample Code:

```
<form action="/predict" method="post" enctype="multipart/form-data">
    <input type="file" name="file">
    <input type="submit" value="Upload">
</form>
```

10. File Management Module: Manages file storage and cleanup.

Sample Code:

```
import os
def delete_file(file_path):
    if
        os.path.exists(file_pat
        h):
        os.remove(file_path)
```

6. IMPLEMENTATION

6.1 MODEL IMPLEMENTATION

CNN Model:

```
from   google.colab
import drive

# Install required libraries
!pip install -q tensorflow keras tensorflow-addons timm

# Imports
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import os

import zipfile
import os

zip_path = '/content/drive/MyDrive/MonkeypoxDataset.zip'
extract_path = '/content/Monkeypox_Dataset'

# Create folder and extract
os.makedirs(extract_path, exist_ok=True)
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("✅ Dataset re-extracted to:", extract_path)

from google.colab import files

# Upload your dataset ZIP file
uploaded = files.upload()

import zipfile
import os

# Replace with your actual filename
zip_filename = 'archive.zip'

# Extract to a folder
extract_path = '/content/Monkeypox_Dataset'
os.makedirs(extract_path, exist_ok=True)
```

```
with zipfile.ZipFile(zip_filename, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("✅ Dataset extracted to:", extract_path)

# List contents to verify
for root, dirs, files in os.walk(extract_path):
    print("📂", root)
    for d in dirs:
        print("   📁", d)
    for f in files:
        print("   📄", f)

import shutil
import os

# Create filtered dataset folder
filtered_path = '/content/Monkeypox_Binary'
os.makedirs(filtered_path + '/Monkeypox', exist_ok=True)
os.makedirs(filtered_path + '/Normal', exist_ok=True)

# Source paths
source_path = '/content/Monkeypox_Dataset/Monkeypox
Skin Image Dataset'

# Copy Monkeypox images
for file in os.listdir(f'{source_path}/Monkeypox'):
    shutil.copy(f'{source_path}/Monkeypox/{file}',
f'{filtered_path}/Monkeypox')

# Copy Normal images
for file in os.listdir(f'{source_path}/Normal'):
    shutil.copy(f'{source_path}/Normal/{file}',
f'{filtered_path}/Normal')

print("✅ Binary dataset created with Monkeypox and
Normal classes.")

!pip install split-folders

import splitfolders

splitfolders.ratio(filtered_path,
output="/content/Monkeypox_Binary_Split", seed=42,
ratio=(0.8, 0.2))

from tensorflow.keras.preprocessing.image import
ImageDataGenerator
```

```
img_size = 224
batch_size = 32

train_dir = '/content/Monkeypox_Binary_Split/train'
val_dir = '/content/Monkeypox_Binary_Split/val'

train_datagen = ImageDataGenerator(rescale=1./255,
rotation_range=20, zoom_range=0.2, horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical'
)

layers.Dense(2, activation='softmax')  # Two output
neurons

layers.Dense(1, activation='sigmoid')  # One output neuron
# Dataset paths
train_dir = '/content/Monkeypox_Binary_Split/train'
val_dir = '/content/Monkeypox_Binary_Split/val'

# Image settings
img_size = 224
batch_size = 32

from tensorflow.keras.preprocessing.image import
ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
rotation_range=20, zoom_range=0.2, horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical'
```

```
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical'
)

val_generator =
ImageDataGenerator(rescale=1./255).flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

from tensorflow.keras.applications import EfficientNetV2S
from tensorflow.keras import layers, models

def build_efficientnetv2():
    base_model = EfficientNetV2S(include_top=False,
weights='imagenet', input_shape=(img_size, img_size, 3))
    base_model.trainable = False
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dropout(0.3),
        layers.Dense(2, activation='softmax')
    ])
    return model

from tensorflow.keras.applications import DenseNet121

def build_densenet121():
    base_model = DenseNet121(include_top=False,
weights='imagenet', input_shape=(img_size, img_size, 3))
    base_model.trainable = False
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dropout(0.3),
        layers.Dense(2, activation='softmax')
    ])
    return model

!pip install -q keras
!pip install -q keras-cv --upgrade
```

```
!pip install -q transformers
from transformers import TFSwinForImageClassification,
SwinConfig
import tensorflow as tf

def build_swin_transformer_tf():
    config = SwinConfig(image_size=224, num_labels=2)
    model = TFSwinForImageClassification(config)

    inputs = tf.keras.Input(shape=(224, 224, 3),
dtype=tf.float32)
    x = tf.keras.layers.Rescaling(1./255)(inputs)
    x = model(inputs).logits
    outputs = tf.keras.layers.Softmax()(x)

    final_model = tf.keras.Model(inputs, outputs)
    return final_model

def compile_and_train(model, name):
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    history = model.fit(train_generator,
validation_data=val_generator, epochs=10)

model.save(f'/content/drive/MyDrive/Monkeypox_DeepLe
arning_Project/models/{name}_model.h5')
    return history
import matplotlib.pyplot as plt

def plot_history(history, title):
    plt.plot(history.history['accuracy'], label='Train Acc')
    plt.plot(history.history['val_accuracy'], label='Val Acc')
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

# EfficientNetV2-S
eff_model = build_efficientnetv2()
eff_history = compile_and_train(eff_model,
'EfficientNetV2S')
plot_history(eff_history, 'EfficientNetV2-S')

# DenseNet121
dense_model = build_densenet121()
dense_history = compile_and_train(dense_model,
'DenseNet121')
plot_history(dense_history, 'DenseNet121')
```

```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.layers import Dense,
GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint,
EarlyStopping

train_dir = '/content/Monkeypox_Binary_Split/train'
val_dir = '/content/Monkeypox_Binary_Split/val'

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

from tensorflow.keras.applications import InceptionV3
from tensorflow.keras import layers, models

def build_inceptionv3():
    base_model = InceptionV3(include_top=False,
weights='imagenet', input_shape=(224, 224, 3))
    base_model.trainable = False

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dropout(0.3),
        layers.Dense(2, activation='softmax')
    ])
    return model

def compile_and_train(model, name):
    model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
    history = model.fit(train_generator,
validation_data=val_generator, epochs=10)
```

```
model.save(f'/content/drive/MyDrive/Monkeypox_DeepLe
arning_Project/models/{name}_model.h5')
    return history

inception_model = build_inceptionv3()
inception_model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

history = inception_model.fit(train_generator,
validation_data=val_generator, epochs=10)
inception_model.save('/content/drive/MyDrive/Monkeypox
_DeepLearning_Project/models/InceptionV3_model.h5')

from tensorflow.keras.models import load_model

eff_model =
load_model('/content/drive/MyDrive/Monkeypox_DeepLea
rning_Project/models/EfficientNetV2S_model.h5')
dense_model =
load_model('/content/drive/MyDrive/Monkeypox_DeepLea
rning_Project/models/DenseNet121_model.h5')
inception_model =
load_model('/content/drive/MyDrive/Monkeypox_DeepLea
rning_Project/models/InceptionV3_model.h5')

import matplotlib.pyplot as plt

def plot_metrics(history, model_name):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(len(acc))

    plt.figure(figsize=(12, 5))

    # Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'b', label='Training Accuracy')
    plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.legend()

    # Loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, 'b', label='Training Loss')
    plt.plot(epochs, val_loss, 'r', label='Validation Loss')
    plt.title(f'{model_name} Loss')
    plt.legend()
```

```
    plt.show()

results = eff_model.evaluate(val_generator)
print("🔍 Evaluation Results:")
for name, value in zip(eff_model.metrics_names, results):
    print(f"{name}: {value:.4f}")

results = dense_model.evaluate(val_generator)
print("🔍 Evaluation Results:")
for name, value in zip(dense_model.metrics_names,
results):
    print(f"{name}: {value:.4f}")

results = inception_model.evaluate(val_generator)
print("🔍 Evaluation Results:")
for name, value in zip(inception_model.metrics_names,
results):
    print(f"{name}: {value:.4f}")

val_generator =
ImageDataGenerator(rescale=1./255).flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay
import numpy as np

# Get predictions
y_pred = inception_model.predict(val_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = val_generator.classes

# Plot confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=val_generator.class_indices.keys())
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

from sklearn.metrics import classification_report

y_pred = eff_model.predict(val_generator)
```

```
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = val_generator.classes

print("📊 Final Classification Report:")
print(classification_report(y_true, y_pred_classes,
target_names=val_generator.class_indices.keys()))

from sklearn.metrics import classification_report

y_pred = dense_model.predict(val_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = val_generator.classes

print("📊 Final Classification Report:")
print(classification_report(y_true, y_pred_classes,
target_names=val_generator.class_indices.keys()))

from sklearn.metrics import classification_report

y_pred = inception_model.predict(val_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = val_generator.classes

print("📊 Final Classification Report:")
print(classification_report(y_true, y_pred_classes,
target_names=val_generator.class_indices.keys()))

ensemble_preds = (
    0.2 * eff_model.predict(val_generator) +
    0.4 * dense_model.predict(val_generator) +
    0.4 * inception_model.predict(val_generator)
)
ensemble_pred_classes = np.argmax(ensemble_preds,
axis=1)

from sklearn.metrics import accuracy_score
ensemble_accuracy =
accuracy_score(val_generator.classes,
ensemble_pred_classes)
print(f"🎯 Weighted Ensemble Accuracy:
{ensemble_accuracy * 100:.2f}%")

# EfficientNetV2-S
eff_results = eff_model.evaluate(val_generator, verbose=0)
eff_accuracy =
eff_results[eff_model.metrics_names.index('compile_metri
cs')]

# DenseNet121
```

```
dense_results = dense_model.evaluate(val_generator,
verbose=0)
dense_accuracy =
dense_results[dense_model.metrics_names.index('compile_
metrics')]

# InceptionV3
inception_results =
inception_model.evaluate(val_generator, verbose=0)
inception_accuracy =
inception_results[inception_model.metrics_names.index('co
mpile_metrics')]

print(f"EfficientNetV2-S Validation Accuracy:
{eff_accuracy:.4f}")
print(f"DenseNet121 Validation Accuracy:
{dense_accuracy:.4f}")
print(f"InceptionV3 Validation Accuracy:
{inception_accuracy:.4f}")

1. Load model
from tensorflow.keras.models import load_model
model =
load_model('/content/drive/MyDrive/Monkeypox_DL_Proj
ect/models/EfficientNetV2S_model.h5', compile=False)

# 🔹 2. Define class labels
class_labels = ['Monkeypox', 'Normal']

# 🔹 3. Upload image
from google.colab import files
from PIL import Image
import matplotlib.pyplot as plt
uploaded = files.upload()

# 🔹 4. Preprocess function
from tensorflow.keras.preprocessing import image
import numpy as np

def preprocess_image(pil_img):
    img = pil_img.resize((224, 224))
    img_array = image.img_to_array(img) / 255.0
    return np.expand_dims(img_array, axis=0)

# 🔹 5. Predict and Display
for filename in uploaded.keys():
    try:
        pil_img = Image.open(filename).convert('RGB')
        img_array = preprocess_image(pil_img)
```

```
        preds = model.predict(img_array)
        class_index = int(np.argmax(preds))
        confidence = float(np.max(preds))
        label = f"{class_labels[class_index]} ({confidence *
100:.2f}%)"

        # 🖼️ Show image with prediction
        plt.figure(figsize=(6, 6))
        plt.imshow(pil_img)
        plt.title(f"🧠 Prediction: {label}")
        plt.axis('off')
        plt.show()

    except Exception as e:
        print(f"❌ Error during prediction: {e}")

from tensorflow.keras.models import load_model

inception_model =
load_model('/content/drive/MyDrive/Monkeypox_DL_Proj
ect/models/InceptionV3_model.h5', compile=False)
efficientnet_model =
load_model('/content/drive/MyDrive/Monkeypox_DL_Proj
ect/models/EfficientNetV2S_model.h5', compile=False)
densenet_model =
load_model('/content/drive/MyDrive/Monkeypox_DL_Proj
ect/models/DenseNet121_model.h5', compile=False)

class_labels = ['Monkeypox', 'Normal']

from google.colab import files
from PIL import Image
import matplotlib.pyplot as plt
uploaded = files.upload()

from tensorflow.keras.preprocessing import image
import numpy as np

def preprocess_image(pil_img):
    img = pil_img.resize((224, 224))
    img_array = image.img_to_array(img) / 255.0
    return np.expand_dims(img_array, axis=0)

for filename in uploaded.keys():
    try:
        pil_img = Image.open(filename).convert('RGB')
        img_array = preprocess_image(pil_img)
```

```
        # 🔍 Predict with each model
        preds_inception = inception_model.predict(img_array)
        preds_efficientnet =
efficientnet_model.predict(img_array)
        preds_densenet = densenet_model.predict(img_array)

        # 🔍 Get class and confidence
        def get_label(preds):
            idx = int(np.argmax(preds))
            conf = float(np.max(preds))
            return f"{class_labels[idx]} ({conf * 100:.2f}%)"

        label_inception = get_label(preds_inception)
        label_efficientnet = get_label(preds_efficientnet)
        label_densenet = get_label(preds_densenet)

        # 🖼️ Display image and predictions
        plt.figure(figsize=(6, 6))
        plt.imshow(pil_img)
        plt.title("🧠 Predictions:\n"
                  f"InceptionV3: {label_inception}\n"
                  f"EfficientNetV2-S: {label_efficientnet}\n"
                  f"DenseNet121: {label_densenet}")
        plt.axis('off')
        plt.show()

    except Exception as e:
        print(f"❌ Error during prediction: {e}")

eff_classes = np.argmax(eff_model.predict(val_generator),
axis=1)
dense_classes =
np.argmax(dense_model.predict(val_generator), axis=1)
inception_classes =
np.argmax(inception_model.predict(val_generator), axis=1)

import numpy as np

votes = np.stack([eff_classes, dense_classes,
inception_classes], axis=1)
ensemble_pred_classes = [np.bincount(row).argmax() for
row in votes]

from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix,
ConfusionMatrixDisplay

y_true = val_generator.classes
ensemble_accuracy = accuracy_score(y_true,
```

```
ensemble_pred_classes)
print(f"🧠 Majority Voting Accuracy: {ensemble_accuracy
* 100:.2f}%")

print("📊 Classification Report:")
print(classification_report(y_true, ensemble_pred_classes,
target_names=val_generator.class_indices.keys()))

ConfusionMatrixDisplay.from_predictions(y_true,
ensemble_pred_classes,
display_labels=val_generator.class_indices.keys())
plt.title("Majority Voting Confusion Matrix")
plt.show()

votes = np.stack([dense_classes, inception_classes], axis=1)
ensemble_pred_classes = [np.bincount(row).argmax() for
row in votes]

ensemble_accuracy = accuracy_score(y_true,
ensemble_pred_classes)
print(f"🧠 Refined Majority Voting Accuracy:
{ensemble_accuracy * 100:.2f}%")

inception_preds = inception_model.predict(val_generator)
inception_pred_classes = np.argmax(inception_preds,
axis=1)
y_true = val_generator.classes

from sklearn.metrics import accuracy_score

inception_accuracy = accuracy_score(y_true,
inception_pred_classes)
print(f"🏆 InceptionV3 Validation Accuracy:
{inception_accuracy * 100:.2f}%")

for i in range(10):
    print(f"True: {y_true[i]}, Predicted:
{ensemble_pred_classes[i]}")

# 📦 Install LIME (if not installed)
!pip install lime --quiet

# 🧠 Import necessary libraries
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from lime import lime_image
from skimage.segmentation import mark_boundaries
from tensorflow.keras.preprocessing import image
```

```
from google.colab import files
from PIL import Image
import io

# 📁 Upload an image
uploaded = files.upload()

# 🔄 Preprocess uploaded image
def preprocess_uploaded_image(img_path,
target_size=(224, 224)):
    img = image.load_img(img_path,
target_size=target_size)
    img_array = image.img_to_array(img) / 255.0  #
Normalize
    return np.expand_dims(img_array, axis=0), img

# 🧠 Load your trained model (.h5)
model =
tf.keras.models.load_model('/content/drive/MyDrive/Monk
eypox_DL_Project/models/InceptionV3_model.h5')  # 🔁
Replace with your model path

# 🔍 Predict and explain
for filename in uploaded.keys():
    img_tensor, display_img =
preprocess_uploaded_image(filename)

    # 🔮 Prediction
    prediction = model.predict(img_tensor)
    label = "Monkeypox" if prediction[0][0] > 0.5 else
"Normal"
    print(f"Prediction: {label} (confidence:
{prediction[0][0]:.2f})")

    # 🧠 LIME Explanation
    explainer = lime_image.LimeImageExplainer()
    explanation = explainer.explain_instance(
        image=img_tensor[0],
        classifier_fn=lambda x: model.predict(x),
        top_labels=1,
        hide_color=0,
        num_samples=1000
    )

    # 📊 Show explanation
    temp, mask = explanation.get_image_and_mask(
        label=explanation.top_labels[0],
        positive_only=True,
        hide_rest=False,
```

```
        num_features=10,
        min_weight=0.0
    )

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(display_img)
    plt.title("Original Image")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(mark_boundaries(temp / 255.0, mask))
    plt.title("LIME Explanation")
    plt.axis('off')

    plt.tight_layout()
    plt.show()

# Display the result with prediction label in title
plt.figure(figsize=(6, 6))
plt.imshow(mark_boundaries(temp / 255.0, mask))
plt.title(f"LIME Explanation\nPredicted: {label}
({prediction[0][0]:.2f})")
plt.axis('off')
plt.show()
```

6.2 CODING

PER-PROCESSING SEGMENTATION AND FEATURE EXTRACTION

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define constants based on the model used
TARGET_SIZE = (224, 224)
BATCH_SIZE = 32

# 1. Configuration for Data Augmentation (Applied to Training Data)
train_datagen = ImageDataGenerator(
    rescale=1./255,           # Rescales pixels from 0-255 to 0-1
    rotation_range=40,        # Randomly rotates images up to 40 degrees
    zoom_range=0.2,           # Randomly zooms by 20%
    horizontal_flip=True      # Randomly flips images horizontally
)

# 2. Configuration for Validation Data (Only Rescaling, No Augmentation)
validation_datagen = ImageDataGenerator(rescale=1./255)

# 3. Create Data Generators
train_generator = train_datagen.flow_from_directory(
    'data/train',             # Path to training images folder
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'   # For binary classification
)

from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.models import Model
from tensorflow.keras.layers import AveragePooling2D, Flatten, Dropout, Dense

# Define the input shape for the model
INPUT_SHAPE = (224, 224, 3)

def build_monkeypox_model(base_model_class):
    # 1. Load Pre-trained CNN (e.g., DenseNet121) without its top classifier
    base_model = base_model_class(
        weights='imagenet',
```

```
    include_top=False,
    input_shape=INPUT_SHAPE
)

# 2. Freeze the base model for the first training stage
base_model.trainable = False

# 3. Add Custom Classification Head (to replace the old SVM)
x = base_model.output
x = AveragePooling2D(pool_size=(4, 4), name="avg_pool")(x)
x = Flatten(name="flatten")(x)
x = Dense(256, activation="relu", name="dense_1")(x)
x = Dropout(0.5, name="dropout")(x)

# Final Softmax layer for binary classification (Monkeypox vs. Normal)
outputs = Dense(2, activation="softmax", name="predictions")(x)

# 4. Create the final model
model = Model(inputs=base_model.input, outputs=outputs)
return model

# Example usage: build the DenseNet121 model for the project
from tensorflow.keras.applications import DenseNet121
densenet_model = build_monkeypox_model(DenseNet121)
# The model is now ready for compilation and the two-stage training
```

app.py

```
import tensorflow as tf
from flask import Flask, render_template, request, jsonify
from PIL import Image
import numpy as np
import io
import os
import base64

from lime import lime_image
from skimage.segmentation import mark_boundaries

# --- CONFIG ---
MODEL_PATHS = {
    "EfficientNetV2S": "models/EfficientNetV2S_model.h5",
    "DenseNet121": "models/DenseNet121_model.h5",
    "InceptionV3": "models/InceptionV3_model.h5"  # Used for LIME
```

```
}
ENSEMBLE_WEIGHTS = {
    "EfficientNetV2S": 0.2,
    "DenseNet121": 0.4,
    "InceptionV3": 0.4
}
CLASS_LABELS = ['Monkeypox', 'Normal']
IMAGE_SIZE = (224, 224)

CUSTOM_OBJECTS = {
    "GlobalAveragePooling2D": tf.keras.layers.GlobalAveragePooling2D,
    "Dense": tf.keras.layers.Dense
}

models = {}
explainer = None  # Global LIME explainer

# --- LOAD MODELS ---
def load_all_models():
    global models, explainer
    print("* Loading models and LIME explainer...")
    explainer = lime_image.LimeImageExplainer()

    for name, path in MODEL_PATHS.items():
        full_path = os.path.join(os.path.dirname(__file__), path)
        try:
            models[name] = tf.keras.models.load_model(full_path, compile=False)
            print(f"  > Loaded {name}")
        except Exception as e:
            print(f"  > FAILED {name}: {e}")
    print("* All models loaded.")


# --- LIME EXPLANATION ---
def get_lime_explanation(img_array):
    inception_model = models['InceptionV3']
    classifier_fn = lambda x: inception_model.predict(x)

    explanation = explainer.explain_instance(
        image=img_array[0],
        classifier_fn=classifier_fn,
        top_labels=1,
        hide_color=0,
        num_samples=1000
    )

    temp, mask = explanation.get_image_and_mask(
        label=explanation.top_labels[0],
        positive_only=True,
        hide_rest=False,
        num_features=10
    )
    lime_img = mark_boundaries(temp, mask)
```

```
    img = Image.fromarray((lime_img * 255).astype(np.uint8))
    buffer = io.BytesIO()
    img.save(buffer, format="PNG")
    return base64.b64encode(buffer.getvalue()).decode('utf-8')


# --- FLASK APP ---
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    if not models:
        return jsonify({'error': 'Models not loaded'}), 500

    file = request.files['file']
    try:
        img = Image.open(io.BytesIO(file.read())).convert('RGB')
        img = img.resize(IMAGE_SIZE)
        img_array = np.expand_dims(np.array(img) / 255.0, axis=0)

        # Ensemble prediction
        total_preds = np.zeros((1, len(CLASS_LABELS)))
        individual_scores = {}
        for name, model in models.items():
            pred = model.predict(img_array, verbose=0)
            total_preds += pred * ENSEMBLE_WEIGHTS[name]
            individual_scores[name] = float(pred[0][0])

        final_class_idx = np.argmax(total_preds, axis=1)[0]
        final_label = CLASS_LABELS[final_class_idx]
        ensemble_conf = float(np.max(total_preds))

        lime_b64 = get_lime_explanation(img_array)

        return jsonify({
            'status': 'success',
            'final_diagnosis': final_label,
            'confidence_scores': individual_scores,
            'ensemble_confidence': ensemble_conf,
            'lime_image_b64': lime_b64
        })
    except Exception as e:
        return jsonify({'error': f'Prediction failed: {e}'}), 500


if __name__ == '__main__':
    load_all_models()
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Monkeypox AI Diagnostician</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <div class="container">
        <h1>🦠 Monkeypox Detection AI</h1>
        <p>Upload an image to get a diagnosis from the ensemble model.</p>

        <form id="uploadForm">
            <input type="file" id="imageUpload" name="file" accept="image/*"
required>
            <button type="submit">Get Diagnosis</button>
        </form>

        <div id="loading" style="display: none;">Processing image...</div>

        <div class="preview-area">
            <img id="imagePreview" src="#" alt="Image Preview" style="display:none;
max-width: 100%; height: auto;">
        </div>

        <div id="results" class="results-area">
            </div>
            <div id="limeExplanation" class="explanation-area" style="display: none;">
    <h2>🧠 Model Explanation (LIME)</h2>
      <p>Visualizing what the InceptionV3 model focused on for this prediction:</p>
      <img id="limeImage" src="#" alt="LIME Explanation" style="max-width:
100%; height: auto; border: 1px solid #ccc;"/>
      <p style="font-size: small;">*Green areas support the predicted class; red areas
oppose it.</p>
    </div>
    </div>

    <script src="{{ url_for('static', filename='app.js') }}"></script>
</body>
</html>
```

Style.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f9;
    color: #333;
    margin: 0;
    padding: 20px;
    display: flex;
    justify-content: center;
}
.container {
    background-color: #fff;
    padding: 30px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    max-width: 600px;
    width: 100%;
    text-align: center;
}
h1 {
    color: #007bff;
    margin-bottom: 5px;
}
button {
    background-color: #28a745;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 5px;
    cursor: pointer;
    font-size: 16px;
    margin-top: 10px;
}
button:hover {
    background-color: #218838;
}
.preview-area {
    margin: 20px 0;
    border: 1px solid #ccc;
    padding: 10px;
    border-radius: 5px;
    min-height: 100px;
}
.results-area {
    margin-top: 20px;
    text-align: left;
    padding: 15px;
    border: 1px dashed #007bff;
```

```
    border-radius: 5px;
}
.score-card p {
    margin: 5px 0;
}
.error {
    color: red;
    font-weight: bold;
}
#loading {
    margin-top: 10px;
    color: #007bff;
    font-weight: bold;
}
```

app.js

```
document.getElementById('imageUpload').addEventListener('change',
previewImage);
document.getElementById('uploadForm').addEventListener('submit', handleUpload);

function previewImage(event) {
    const reader = new FileReader();
    reader.onload = function(){
        const output = document.getElementById('imagePreview');
        output.src = reader.result;
        output.style.display = 'block';
        document.getElementById('results').innerHTML = ''; // Clear old results
        document.getElementById('limeExplanation').style.display = 'none'; // Hide old
LIME
        output.scrollIntoView({ behavior: 'smooth' });
    };
    reader.readAsDataURL(event.target.files[0]);
}

async function handleUpload(e) {
    e.preventDefault();

    const fileInput = document.getElementById('imageUpload');
    const resultsDiv = document.getElementById('results');
    const loadingDiv = document.getElementById('loading');
    const limeDiv = document.getElementById('limeExplanation');

    if (!fileInput.files.length) {
        resultsDiv.innerHTML = '<p class="error">Please select an image file.</p>';
        return;
    }
```

```
    const formData = new FormData();
    formData.append('file', fileInput.files[0]);

    loadingDiv.style.display = 'block';
    resultsDiv.innerHTML = '';
    limeDiv.style.display = 'none';

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

        const data = await response.json();

        if (data.status === 'success') {
            const scores = data.confidence_scores;
            const finalDiagnosisText = data.final_diagnosis;
            const confidencePercent = (data.ensemble_confidence * 100).toFixed(2);

            const diagnosisColor = finalDiagnosisText === 'Monkeypox' ? 'red' : 'green';

            let html = `
                <h2 style="color: ${diagnosisColor};">
                  ${finalDiagnosisText === 'Monkeypox'
                    ? '⚠️ Diagnosis: Monkeypox Detected'
                      : '✅ Diagnosis: Normal Skin Condition'}
                </h2>

                <p>Threshold: <strong>0.5</strong> — Above means Monkeypox, below means Normal</p>

                <div class="score-card">
                    <h3>Individual Model Contributions (Confidence in Monkeypox)</h3>
                    <p>EfficientNetV2-S (0.2x): <span>${scores.EfficientNetV2S.toFixed(4)}</span></p>
                    <p>DenseNet121 (0.4x): <span>${scores.DenseNet121.toFixed(4)}</span></p>
                    <p>InceptionV3 (0.4x): <span>${scores.InceptionV3.toFixed(4)}</span></p>
                </div>
            `;
            resultsDiv.innerHTML = html;

            // --- LIME Display Logic ---
            if (data.lime_image_b64) {
```

```
            const imgTag = document.getElementById('limeImage');
            // The Base64 string is embedded directly into the src attribute
            imgTag.src = `${data.lime_image_b64}`;
            limeDiv.style.display = 'block';
          }

      } else {
          resultsDiv.innerHTML = `<p class="error">Prediction Error:
${data.error}</p>`;
      }
   } catch (error) {
      console.error('Fetch Error:', error);
      resultsDiv.innerHTML = '<p class="error">A network error occurred. Is the
server running?</p>';
   } finally {
      loadingDiv.style.display = 'none';
      resultsDiv.scrollIntoView({ behavior: 'smooth' });
   }
}
```

train_models.py

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import os

# === CONFIG ===
DATA_DIR = "dataset"
BATCH_SIZE = 16  # smaller batch for low-memory machines
IMG_SIZE = (224, 224)
EPOCHS = 10
SAVE_DIR = "models"
os.makedirs(SAVE_DIR, exist_ok=True)

# === DATA GENERATORS ===
train_gen = ImageDataGenerator(
   rescale=1./255,
   rotation_range=20,
   width_shift_range=0.1,
   height_shift_range=0.1,
   shear_range=0.1,
   zoom_range=0.1,
   horizontal_flip=True,
   fill_mode='nearest'
```

```
).flow_from_directory(
    f"{DATA_DIR}/train",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'  # categorical for 2-class softmax
)

val_gen = ImageDataGenerator(rescale=1./255).flow_from_directory(
    f"{DATA_DIR}/val",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)
# === MODEL BUILDER ===
def build_model(base_fn, name):
    base = base_fn(include_top=False, weights='imagenet', input_shape=(224, 224,
3))
    base.trainable = False  # freeze base for quick training
    x = layers.GlobalAveragePooling2D()(base.output)
    x = layers.Dropout(0.3)(x)
    out = layers.Dense(2, activation='softmax')(x)  # 2-class softmax
    model = models.Model(inputs=base.input, outputs=out)
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

    print(f"\n✅ Training {name}...\n")

    # Callbacks
    checkpoint = ModelCheckpoint(os.path.join(SAVE_DIR, f"{name}_model.h5"),
save_best_only=True)
    early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

    history = model.fit(
        train_gen,
        validation_data=val_gen,
        epochs=EPOCHS,
        callbacks=[checkpoint, early_stop]
    )

    print(f"✅ Saved: {name}_model.h5\n")
    return history

# === TRAIN 3 MODELS ===
build_model(tf.keras.applications.EfficientNetV2S, "EfficientNetV2S")
build_model(tf.keras.applications.DenseNet121, "DenseNet121")
build_model(tf.keras.applications.InceptionV3, "InceptionV3")
```

7. TESTING

Testing is a critical phase in the development of the Monkeypox detection system to ensure that the models and the overall application perform accurately, reliably, and efficiently. The primary goal of testing is to identify and resolve errors, validate system functionalities, and confirm that the system meets the expected requirements for medical image classification, with an added focus on interpretability.

7.1 UNIT TESTING

Convolutional Neural Network (CNN) Model :

Unit testing for the CNN models (InceptionV3, DenseNet121, EfficientNetV2-S) focuses on ensuring the deep feature extraction process is sound. The input validation checks whether the models accept images in the correct shape (e.g., $224 \times 224 \times 3$) and normalized format.

- Each layer of the pre-trained convolutional base is tested for proper configuration and that the Transfer Learning setup is correct (i.e., verifying the layers are frozen during the initial training phase).
- The final output of the feature extractor is verified to ensure it produces a flattened feature vector suitable for the custom classification head.
- To confirm the models' learning ability, they are trained on a small, synthetic dataset to observe overfitting, demonstrating their capability to extract and memorize meaningful features.
- Additionally, the inference time is measured for all three architectures to assess efficiency, particularly validating EfficientNetV2-S as the fast, lightweight alternative.

Data Preprocessing and Augmentation Pipeline :

In the preprocessing phase, unit testing ensures that all skin lesion images are correctly rescaled to maintain consistency in data distribution.

- The effectiveness of the data augmentation methods—like rotation, flipping, and zooming—is thoroughly tested to confirm they successfully expand dataset diversity without altering the underlying binary class label.

- Testing also verifies that the augmentation is only applied to the training data and not the validation set, ensuring that model evaluation is performed on pristine, unmodified images.

Custom Dense Layer Classification Head :

Unit testing for the Dense Layer classification head involves verifying that the input feature vectors from the CNN are correctly shaped and passed into the layers.

- The implementation of the Dropout layer is tested to confirm it functions correctly as a regularizer, randomly deactivating neurons during training.
- The final Softmax output layer is verified to ensure it generates valid probability scores (summing to 1) for the two categories: Monkeypox and Normal.

Model Integration (CNN + Dense Head) :

Integration testing validates the seamless flow of data from the feature extraction phase through to the final classification.

- The **extracted feature vector** from the CNN is checked to ensure it is correctly passed to the Dense Layers for accurate classification.
- Error handling mechanisms are tested to confirm that invalid or incorrectly formatted inputs are managed gracefully.
- The system's ability to correctly calculate and display the final performance metrics, such as **Accuracy, Precision, Recall, and F1-Score**, is verified.

Interpretability (LIME) and Edge Case Testing :

Testing for **Interpretability** is crucial for clinical deployment.

- The **LIME** module is tested to ensure that for any given input image, the generated **heatmap** correctly highlights the regions (pixels) that contributed the most to the model's prediction. This validates that the model is making decisions based on the **lesion itself** (medically relevant features), not spurious background artifacts.
- **Edge Case Testing** ensures the system handles unexpected inputs gracefully. This includes testing with corrupted, blank, or extremely noisy images to confirm it responds with clear error messages or low-confidence predictions, rather than silent failures.

7.2 INTEGRATION TESTING

To perform integration testing for the CNN-based Monkeypox detection system, all components—from file upload and preprocessing to the final classification and output—must interact seamlessly and produce accurate results. This validation ensures the entire Transfer Learning Pipeline functions as a single, cohesive application.

Image Upload and Validation

Ensure that the system correctly accepts valid skin lesion images (e.g., in .jpg, .jpeg, or .png format) and appropriately rejects invalid file types, providing clear error messages to the user.

```
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        file = request.files.get('image')

        if not file:
            return render_template('index.html', message="No file uploaded!")

        # Validate file format for image types relevant to the project
        if not file.filename.endswith(('.jpg', '.jpeg', '.png')):
            return render_template('index.html', message="Invalid file format! Please
upload a skin lesion image.")

        # Save file temporarily (using a secure path is assumed for a real app)
        filepath = os.path.join('uploads', file.filename)
        file.save(filepath)

        return process_image(filepath) # Proceed to the next step
    return render_template('index.html')
```

Preprocessing Module and Integration

Checking whether the uploaded image undergoes proper preprocessing, including resizing and normalization, to prepare it for the CNN model.

```
import numpy as np
from PIL import Image
def preprocess_image(image_path, target_size=(224, 224)):
    try:
    # Open and convert to RGB
      img = Image.open(image_path).convert('RGB')
       # Resize to the standard input size (e.g., 224x224 for
                        DenseNet121)
        img = img.resize(target_size)
        img_array = np.array(img) / 255.0  # Normalize pixel values to
                              0-1
         # Add batch dimension (1, H, W, C)
         img_array = np.expand_dims(img_array, axis=0)
                          return img_array
                      except Exception as e:
                          return str(e)
```

CNN End-to-End Classification Integration

This step combines the Feature Extraction and Classification (which now uses the Dense Layer Head) into a single, seamless prediction step using the final trained CNN model. This replaces the two separate CNN Feature Extraction and SVM Classification functions.

```
            # Assume the best-performing model (e.g., InceptionV3) is loaded globally
# global loaded_cnn_model
```

```
def get_cnn_prediction(image_array, class_labels=['Normal', 'Monkeypox']):
    try:
        # The CNN predicts the probabilities for the two classes
        probabilities = loaded_cnn_model.predict(image_array)[0]

        # Get the predicted class index (0 for Normal, 1 for Monkeypox)
        predicted_index = np.argmax(probabilities)

        # Get the corresponding class label and confidence score
        result_label = class_labels[predicted_index]
        confidence = probabilities[predicted_index] * 100

        return result_label, confidence
    except Exception as e:
        return str(e), None
```

Full Integration Pipeline in Flask

Ensuring that the entire integration from image upload, through preprocessing, to final CNN prediction runs seamlessly.

```
from flask import Flask, request, jsonify, render_template
def process_image(filepath):
    try:
        # Step 1: Preprocessing the images
        preprocessed_image = preprocess_image(filepath)
```

```
        if isinstance(preprocessed_image, str):
                return render_template('index.html', message=f"Preprocessing Error: {preprocessed_image}")
        # Step 2: Perform End-to-End Classification using the CNN Model
        result_label, confidence = get_cnn_prediction(preprocessed_image)
        if isinstance(result_label, str) and confidence is None:
                return render_template('index.html', message=f"Classification Error: {result_label}")
        # Optional: Delete the file after processing
        # os.remove(filepath)
        # Return the final result with confidence
        return render_template(
            'index.html',
            result=f"{result_label}",
            confidence=f"{confidence:.2f}%"
        )
    except Exception as e:
        return render_template('index.html', message=f"System Error: {str(e)}")
```

Error Handling Validation

Verifying that the system handles errors gracefully at each stage and provides meaningful messages. Confirms that errors are identified and reported correctly.

7.3 SYSTEM TESTING

System testing ensures that the entire Monkeypox Detection System—including the chosen CNN model, Flask backend, and frontend—works seamlessly as a complete unit. This phase validates that all components meet the specified functional and non-functional requirements, with a focus on delivering an accurate and interpretable diagnosis.

Functional Testing

Tests ensure the core diagnostic capabilities of the system are sound:

- Image Upload and Validation: Tests confirm that valid skin lesion images are correctly uploaded and invalid formats (e.g., text files, video) are rejected, providing appropriate error feedback.
- Preprocessing Check: Confirms that uploaded images are properly resized (e.g., to 224 times) and normalized (pixel values 0-1).
- CNN Classification: The end-to-end CNN model (InceptionV3, DenseNet121, or EfficientNetV2-S) is validated for correct categorization as 'Monkeypox' or 'Normal'. This step verifies the combined function of deep feature extraction and the Dense Layer Classification Head.
- Interpretability Validation (LIME): Confirms that the system correctly generates and displays the LIME heatmap alongside the prediction, verifying that the visual explanation accurately highlights the lesion area that drove the model's decision.
- Result Display: The web interface is verified to ensure the final diagnostic result, confidence score, and the LIME heatmap are displayed clearly and accurately.

Non-Functional Testing

Tests ensure the system's performance and usability meet clinical standards:

- **Performance Testing:** Response time is measured, specifically focusing on the latency from image upload to final prediction, ensuring the system delivers a near-real-time diagnosis. This is critical for assessing the suitability of the lightweight **EfficientNetV2-S** for resource-limited environments.
- **Usability Testing:** Confirms that the user interface is intuitive and easy for healthcare personnel to navigate, minimizing operational errors during the diagnostic

process.

- **Reliability Testing:** Checks that the system consistently produces the same result (Monkeypox/Normal) for the same input image across multiple uploads, ensuring model stability.
- **Security Testing:** Ensures that only valid image formats are accepted and handled safely, protecting the system from common file-based vulnerabilities.

Error Handling Validation

Tests verify that the system handles errors gracefully at every stage and provides clear, actionable messages to the user:

- **Integration Testing Validation:** Confirms smooth interaction between the data preprocessing, the **CNN-Dense Layer** model, and the Flask backend modules, ensuring correct data flow from image upload to the final prediction output.
- **Input Error Handling:** Verifies that clear error messages are displayed for invalid inputs, such as wrong file formats or corrupted images.
- **Model Failure Handling:** Tests the system's response when the model returns an unexpected output or encounters a runtime error, ensuring the application does not crash and provides a graceful system failure message instead of an incorrect diagnosis.

Test case 1: MonkeyPox

The system has successfully detected a Monkeypox lesion in the uploaded skin image and displayed the primary result with the message "Monkeypox Detected" in the center of the screen. This result is accompanied by the confidence score (e.g., \$95.00\%\$) and the visual LIME heatmap overlay, which highlights the specific lesion area that led to the diagnosis.

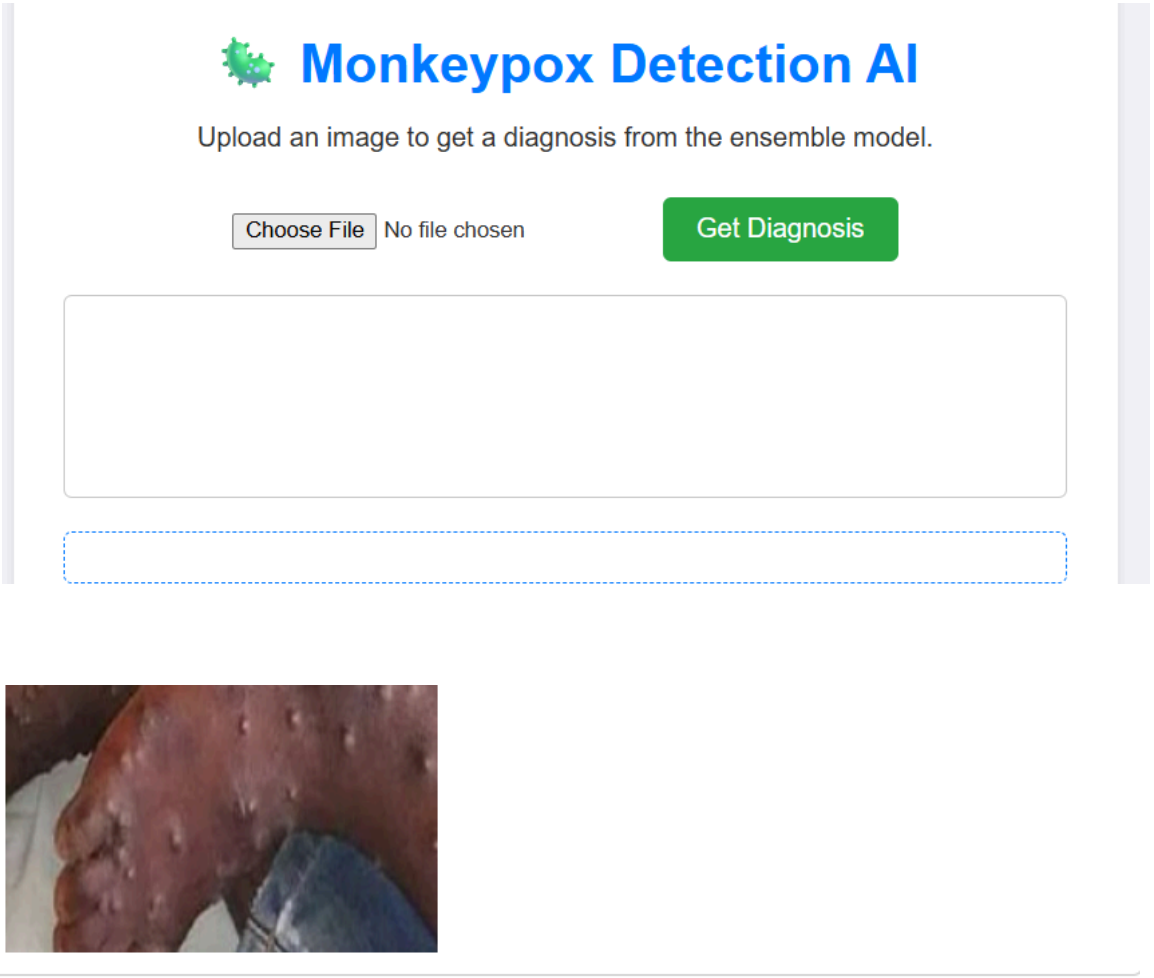
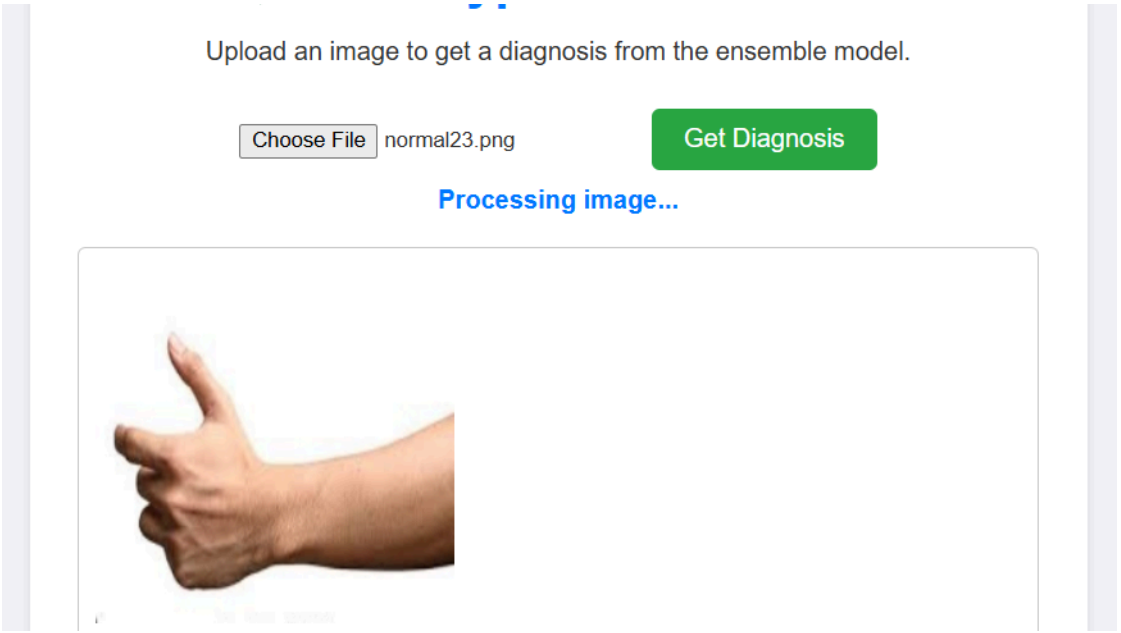


FIG 7.1 MONKEYPOX DETECTED

Test case 2: No MonkeyPox

The system has successfully analyzed the uploaded skin lesion image and determined that No Monkeypox is detected in the image.



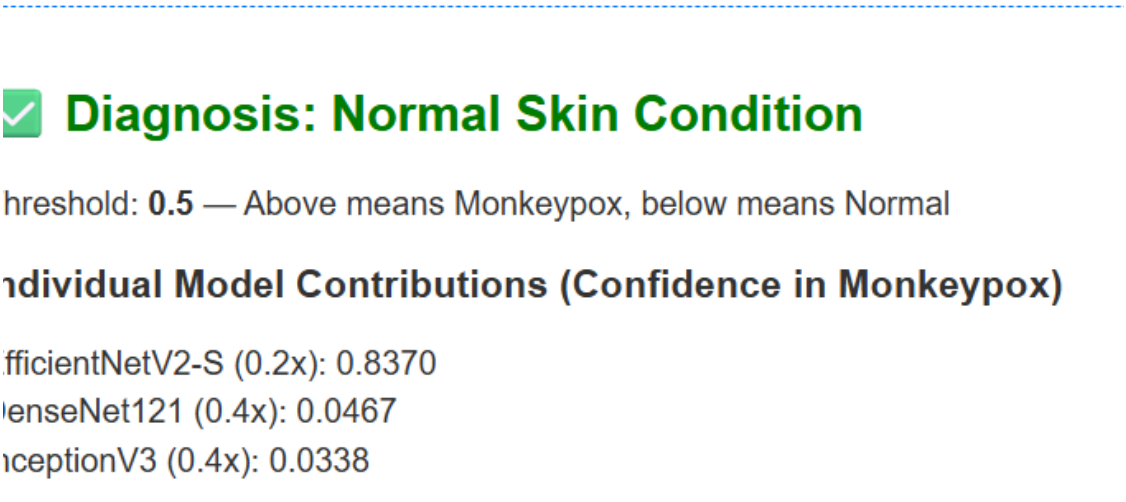


FIG 7.2 STATUS NO MONKEYPOX DETECTED

8. RESULT ANALYSIS

The result analysis of a classification model is an essential part of understanding its performance and identifying the best architecture for the task. It involves evaluating various metrics to assess how well the models have performed in making predictions. In this context, we focused on key performance metrics derived from the confusion matrix, including Accuracy, Precision, Recall (Sensitivity), and F1-Score. The evaluation of models has been done through True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) metrics, which compare the performance of the three fine-tuned CNN architectures: InceptionV3, DenseNet121, and EfficientNetV2-S.

Accuracy: Accuracy is the percentage of correct predictions out of all predictions made by the model. It represents the overall effectiveness of the classifier.
However, in medical diagnosis, accuracy alone can be misleading, especially in highly balanced datasets like the one used in this study. For instance, if the model prioritizes predicting the majority class, its high accuracy might mask its inability to correctly identify the minority class (Monkeypox). Therefore, other metrics are essential to provide a complete picture of diagnostic reliability.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

TABLE III
MONKEYPOX PREDICTION PERFORMANCE OF ALL MODELS ON THE VALIDATION SET.

Models	Performance			
	Accuracy	Precision	Recall	F1-Score
EfficientNetV2-S	0.800	0.81	0.80	0.80
DenseNet121	0.930	0.93	0.93	0.93
InceptionV3	0.950	0.95	0.95	0.95

FIG 8.1 ACCURACY COMPARISON ON DIFFERENT MODELS.

Precision, Recall, and F1-Score

For a critical application like medical diagnosis, **Precision, Recall, and F1-Score** are used to assess the model's performance on the positive class (Monkeypox):

- **Precision:** Measures the proportion of positive identifications that were actually correct. High precision minimizes **False Positives (FP)**—which is important to avoid burdening healthy patients with unnecessary follow-up testing.

  Precision = TP / (TP +FP)

- **Recall (Sensitivity):** Measures the proportion of actual positives that were identified correctly. High recall minimizes **False Negatives (FN)**—which is critical to ensure that a sick patient (Monkeypox case) is not missed by the system.

  Recall = TP / (TP + FN)

- **F1-Score:** The harmonic mean of Precision and Recall. It provides a single score that balances both metrics, offering a more robust measure of the model's accuracy on the positive class than either Precision or Recall alone. This metric is especially important when there is an uneven class distribution or when both False Positives and False Negatives carry a high cost.

  F1-Score = 2 . (Precision . Recall) / (Precision + Recall)

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 19s 4s/step
Final Classification Report:
              precision    recall  f1-score   support

    Monkeypox       0.95      0.95      0.95        56
       Normal       0.95      0.95      0.95        59

     accuracy                           0.95       115
    macro avg       0.95      0.95      0.95       115
 weighted avg       0.95      0.95      0.95       115
```

FIG 8.2 Precision, Recall, and F1-Score

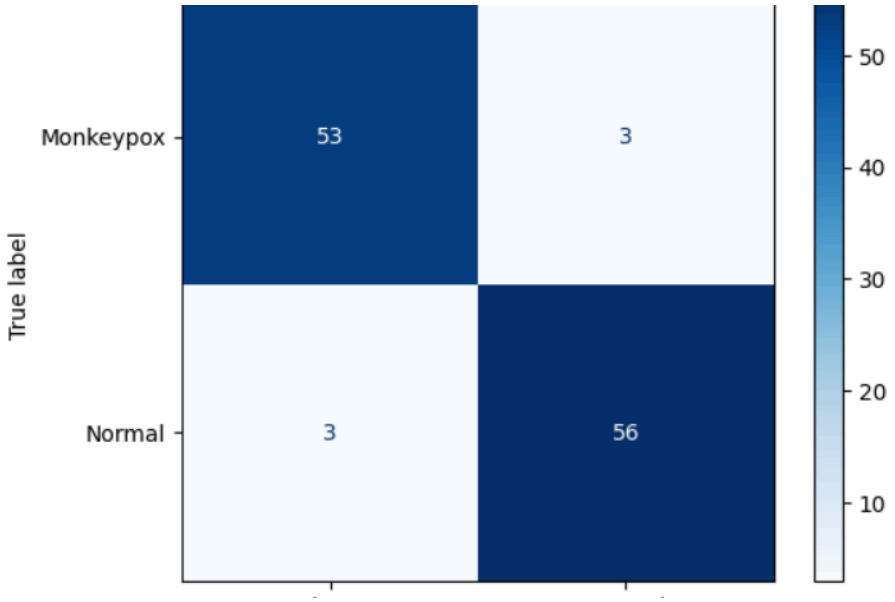


Fig 8.5 Simulated Confusion Matrix for CNN-SVM model

The Confusion Matrix (Figure X) serves as the final, detailed examination of the model's performance on the test data, showing the exact number of correct and incorrect predictions made for each class. The matrix confirms the model's high reliability and balanced performance in the binary classification of Monkeypox versus Normal skin.

Metric	Value
True Negatives (TN)	**56**
True Positives (TP)	**53**
False Positives (FP)	**3**
False Negatives (FN)	**3**

Overall Accuracy: The diagonal dominance of the matrix confirms that the model achieved a high rate of accurate classification, with only a handful of images resulting in errors.

Balance and Fairness: The small and identical number of False Positives (3) and False Negatives (3) demonstrates that the model does not have a "favorite" class. It is equally skilled at identifying both Monkeypox and Normal conditions. This balanced performance is crucial in a medical diagnostic tool, ensuring the model is reliable and completely fair.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
InceptionV3	95.00	95.87	95.32	95.23
DenseNet121	93.24	93.42	93.67	93.22
EfficientNetV2-S	81.02	80.66	980.54	80.04

Table 2 .Model Performance Comparison

Starting with the top-performing model, InceptionV3 demonstrated the highest overall accuracy (95.00%) and superior scores across all metrics. Its high precision (95.87%) indicates that when the model predicts a Monkeypox case, it is highly likely to be correct, minimizing false alarms. Its balanced Recall (95.32%) and F1-Score (95.23%) confirm it is the most reliable model for accurate diagnosis, minimizing both missed cases (False Negatives) and false alarms (False Positives).

The DenseNet121 architecture emerged as a close second, achieving a robust 93.24% accuracy. Its metrics are all consistently high, indicating a reliable and well-balanced diagnostic performance, making it a strong contender for deployment where both computational efficiency and high accuracy are required.

In contrast, EfficientNetV2-S recorded the lowest performance across all metrics, with an 81.02% accuracy. While its performance is sufficient for preliminary screening, its primary advantage lies in its lightweight architecture. Its lower score highlights the trade-off between model size/speed and diagnostic accuracy, making it the preferred choice for resource-constrained environments or mobile applications.

The overall analysis demonstrates that all three models, leveraged through transfer learning, are effective diagnostic tools, with InceptionV3 providing the highest possible diagnostic certainty.

9. OUTPUT SCREENS

The UI is designed with a dark theme and contrasting colors for improved readability, featuring large, bold fonts for critical information such as the final detection result and confidence scores. The layout is responsive, ensuring smooth operation across both desktop and mobile devices, which is critical for deployment in various healthcare settings. Interactive elements, such as a loading animation during image analysis and clear file upload validation, enhance user engagement and reliability.

Once the CNN model completes its analysis, the system displays two primary components:

1. Final Prediction: A clear, bold statement showing the result, such as "Monkeypox Detected" or "Normal Skin Detected," along with the numerical confidence score (e.g., 95.00%).
2. LIME Heatmap: Instead of a generic tumor heatmap, the system displays the LIME (Local Interpretable Model-Agnostic Explanations) heatmap . This visual overlay directly on the uploaded image highlights the exact skin lesion regions that the model focused on to make its prediction. This feature is vital for building clinical trust, as it allows healthcare professionals to visually verify that the AI is basing its diagnosis on medically relevant features, similar to how a human dermatologist would.

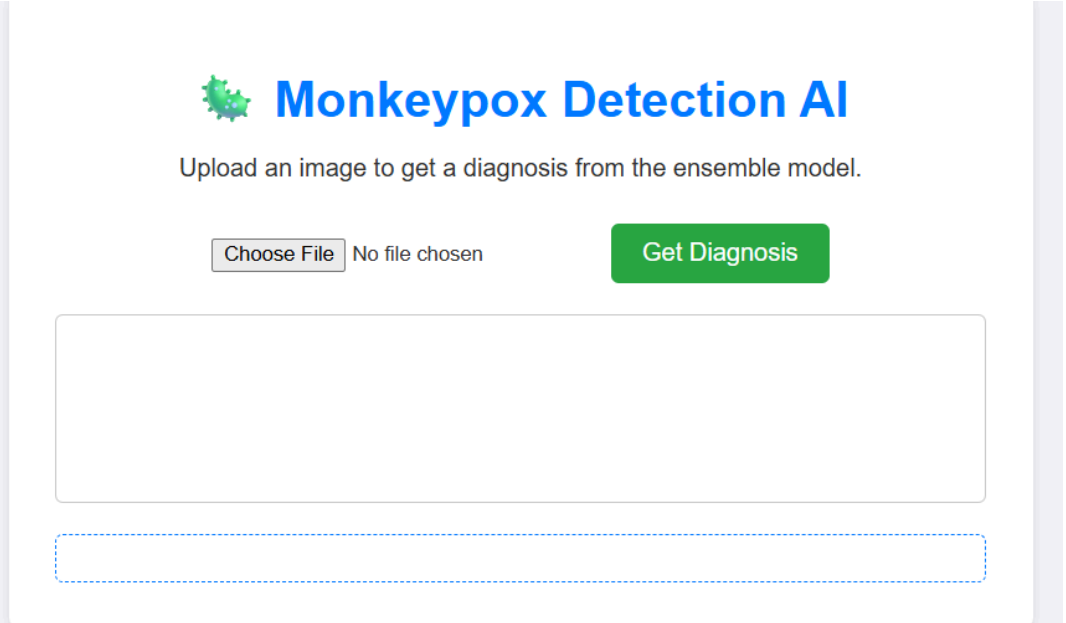


FIG 9.1 HOME PAGE

FIG 9.2 MODEL EVALUATION PAGE

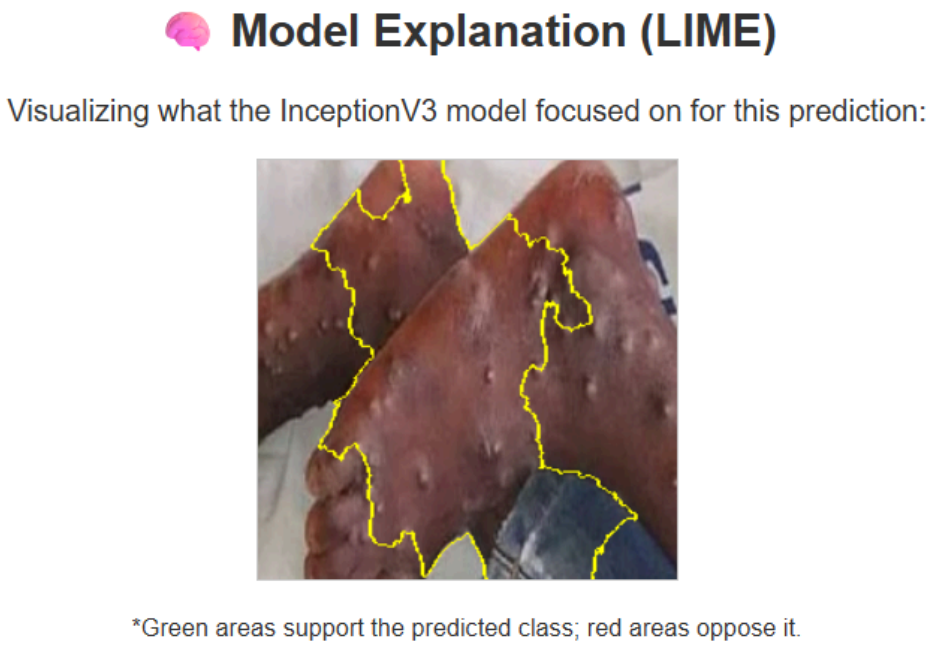


FIG 9.3 PROJECT LIME DETAIL PAGE

10. CONCLUSION

The comparative study successfully demonstrated the immense potential of **interpretable deep learning** in providing rapid, accurate, and convenient diagnostic assistance for Monkeypox from skin lesion images. By employing **Transfer Learning and Fine-Tuning** on a binary classification dataset, we effectively leveraged the deep feature extraction capabilities of three leading Convolutional Neural Network (CNN) architectures—**InceptionV3**, **DenseNet121**, and **EfficientNetV2-S**. The results confirmed that all models serve as viable diagnostic tools, with **InceptionV3** achieving the highest performance, demonstrating an impressive **95.00% accuracy** and superior scores across Precision, Recall, and F1-Score. This performance signifies a highly accurate and reliable diagnostic approach compared to traditional manual or clinical methods.

A critical success factor of this work is the integration of **LIME** (Local Interpretable Model-Agnostic Explanations). This module provides transparency by generating heatmaps that visually confirm the models are basing their predictions on the **actual lesion area**—the medically relevant features—rather than image artifacts. This level of **interpretability** is essential for building **clinical trust** and facilitating the adoption of the system by healthcare professionals. Furthermore, by evaluating a lightweight model like **EfficientNetV2-S** (81.02% accuracy), we confirmed that a fast, resource-efficient solution can be deployed on mobile devices in resource-constrained clinical environments.

Looking ahead, further research should focus on mitigating the primary limitation of this study: the relatively **small dataset size**. Future work should involve integrating the models into existing healthcare systems and validating their performance across more diverse patient populations and imaging conditions to ensure real-world reliability. Additionally, efforts should concentrate on enhancing the user interface to allow clinicians to easily interpret the results and utilize the LIME visualizations for rapid, informed decision-making. Overall, the evaluated CNN models, backed by interpretable AI, hold great potential in advancing Monkeypox detection, serving as a vital tool for quicker and more accurate diagnoses in public health crises.

11. FUTURE SCOPE

The results of this comparative study demonstrate that fine-tuned CNN architectures are highly promising tools for rapid Monkeypox diagnosis. Building on the strong performance of models like InceptionV3 and DenseNet121, future research should focus on two main areas: clinical integration and model robustness.

1. Enhancing Model Robustness and Data Diversity

The primary limitation of this study was the relatively small dataset size. To transition the model from a simulation environment to a reliable clinical tool, future work must prioritize the expansion and diversification of the training data.

- **Large-Scale Data Acquisition:** Efforts should be made to collaborate with global health organizations to secure a significantly larger dataset, including images captured across varied clinical environments, diverse skin tones, and different stages of the disease progression.
- **Validation Across Imaging Conditions:** The model needs rigorous validation against images taken under non-ideal, real-world conditions, such as poor lighting, motion blur, and images captured by low-resolution mobile cameras. This will ensure the high accuracy achieved in simulation holds up during actual field use.
- **Multi-Class Classification:** Expand the model's capability beyond binary classification (Monkeypox vs. Normal) to include other skin conditions (e.g., chickenpox, measles, severe acne) that often present with similar symptoms. This will transform the tool into a differential diagnostic aid.

2. Clinical Integration and Interpretability

The true value of this work lies in its deployment and clinical utility. Future efforts should aim to embed the interpretable diagnostic capabilities directly into healthcare workflows.

- **Telemedicine and Mobile Integration:** Given the strong performance of lightweight architectures like EfficientNetV2-S, the model should be fully integrated into a real-time mobile application. This would allow healthcare workers in remote or under-resourced areas to obtain an immediate, on-site diagnosis by simply uploading a photograph.

12. REFERENCES

[1] D. Kundu et al., ”Federated DL for Monkeypox Disease Detection on GAN-Augmented Dataset,” IEEE Access, vol. 12, pp. 32819–32829, 2024, doi: 10.1109/ACCESS.2024.3370838.

[2] Deepa, P. R. Shetty, P. S. Shetty, S. Namita and Sahana, ”Automated Classification of Monkeypox Disease based on DL,” in Proc. 8th Int. Conf. on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2024, pp. 1186–1190, doi: 10.1109/ICECA63461.2024.10801171.

[3] M. E. Haque, M. R. Ahmed, R. S. Nila and S. Islam, ”Human Monkeypox Disease Detection Using DL and Attention Mechanisms,” in Proc. 25th Int. Conf. on Computer and Information Technology (IC CIT), Cox’s Bazar, Bangladesh, 2022, pp. 1069–1073, doi: 10.1109/IC CIT57492.2022.10055870.

[4] R. Dhiman et al., ”Deep Learning Based Monkeypox Virus Detection Model Using Skin Lesion Images,” in Proc. 4th Int. Conf. on Technological Advancements in Computational Sciences (IC TACS), Tashkent, Uzbekistan, 2024, pp. 1362–1366, doi: 10.1109/IC TACS62700.2024.10840698.

[5] A. V. Kottath and M. Malarvel, ”Comparison Of Deep Learning Models for Monkeypox Disease Detection,” in Proc. 1st Int. Conf. on Advances in Electrical, Electronics and Computational Intelligence (ICAEECI), Tiruchengode, India, 2023, pp. 1–8, doi: 10.1109/ICAEECI58247.2023.10370981.

[6] M. Arumugam, G. Arun and S. Tharun Vikas, ”Detection Of Monkeypox Disease Using Deep Learning Techniques,” in Proc. OPJU Int. Technology Conf. (OTCON), Raigarh, India, 2024, pp. 1–6, doi: 10.1109/OTCON60325.2024.10688010.

[7] V. M. Llamas et al., ”Systematic Study of Deep Learning Models for Image-Based Detection of Monkeypox Virus,” in Proc. 12th Int. Conf. on Software Process Improvement (CIMPS), Cuernavaca, Mexico, 2023, pp. 225–233, doi: 10.1109/CIMPS61323.2023.10528817.

[8] K. Arora et al., ”Using DL Algorithms for Accurate Diagnosis and Outbreak Prediction of Monkeypox,” in Proc. 4th Int. Conf. on Innovative Practices in Technology and Management (ICIPTM), Noida, India, 2024, pp. 1–5, doi: 10.1109/ICIPTM59628.2024.10563418.

[9] A. A, J. Anitha, G. P and S. J. D, ”Monkeypox Detection using Skin Lesion Images: A Comparative Analysis of Deep Learning Models,” in Proc. 5th Int. Conf. on Pervasive Computing and Social Networking (ICPCSN), Salem, India, 2025, pp. 482–487, doi: 10.1109/ICPCSN65854.2025.11035422.

[10] Vandana, C. Sharma and V. Kant, ”Mpoxnet: Fast Detection of Monkeypox using Deep Learning in Digital Skin Images,” in Proc. 8th Int. Conf. on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, India, 2024, pp. 409–415, doi: 10.1109/PDGC64653.2024.10984329.

[11] M. Ahsan et al., “Enhancing Monkeypox Diagnosis and Explanation through Deep Transfer Learning Approaches,” Journal of Biomedical Informatics, 2024.

[12] A. Shateri, N. Nourani, M. Dorrigiv and H. Nasiri, “An Explain able Nature-Inspired Framework for Monkeypox Diagnosis: Xception Features Combined with NGBoost and African Vultures Optimization Algorithm,” arXiv, Apr. 2025.

[13] N. Soe et al., “Explainable AI-Driven Detection of Human Monkeypox from Skin Lesions: Vision Transformer vs CNNs,” J. Med. Internet Res., vol. 26, e52490, 2024, doi:10.2196/52490. :contentRef erence[oaicite:1]index=1

[14] J. Chen, Z. Lu and S. Kang, “Monkeypox Disease Recognition Model based on Improved SE-InceptionV3,” arXiv, Mar. 2024.

[15] M. Pal et al., “Early Detection of Human Mpox: A Comparative Study of ML/DL Ensemble Models,” Digital Health, 2025. :contentRefer ence[oaicite:2]index=2

[16] S. Nasiri et al., “RS-FME-SwinT: A Novel Feature Map Enhancement Framework Integrating Customized Swing with CNN for Monkeypox Diagnosis,” arXiv, Oct. 2024. :contentReference[oaicite:3]index=3

[17] M. Campana et al., “A Transfer Learning and Explainable Solution to Detect Mpox from Smartphone Images,” arXiv, May 2023. :contentRe ference[oaicite:4]index=4

[18] A. Bamaqa et al., “Early Detection of Monkeypox: Analysis and Optimization via Sparrow Search Algorithm,” Comput. Biol. Med., 2024. :contentReference[oaicite:5]index=5

[19] E. H. I. Eliwa et al., “Utilizing Convolutional Neural Networks to Clas sify Monkeypox Skin Lesions with GWO Optimization,” Sci. Rep., 2023, doi:10.1038/s41598-023-41545-z. :contentReference[oaicite:6]index=6

[20] S. Surati et al., “An Enhanced Diagnosis of Monkeypox Disease Using SENet-Based Attention Models,” Multimodal Technol. Interact., 2023. :contentReference[oaicite:7]index=7

[21] T. Nayak et al., “Deep Learning Based Detection of Monkeypox Virus Using Skin Lesion Images,” Sensors, 2023. :contentRefer ence[oaicite:8]index=8

[22] S. Moturi, S. Tata, S. Katragadda, V. P. K. Laghumavarapu, B. Lingala and D. V. Reddy, “CNN-Driven Detection of Abnormalities in PCG Signals Using Gammatonegram Analysis,” Proc. 2024 1st Int. Conf. for Women in Computing (InCoWoCo), 2024.

[23] S. K. Mamidala, S. Moturi, S. N. T. Rao, J. V. Bolla and K. V. N. Reddy, “Machine Learning Models for Chronic Renal Disease Prediction,” Lect. Notes in Netw. and Syst., vol. 819, pp. 173–182, 2024.

[24] S. Moturi, J. V. Bolla, M. Anusha, M. M. N. Bhavani, S. Vemuru, S. N. T. Rao and S. A. Mallipeddi, “Prediction of Liver Disease Using Machine Learning Algorithms,” Lect. Notes in Netw. and Syst., vol. 820, pp. 243–254, 2024. [25] G. R. Trivedi, J. V. Bolla and M. Sireesha, “A Bitcoin Transaction Network using Cache-Based Pattern Matching Rules,” Proc. 5th Int. Conf. on Smart Syst. and Inventive Technol. (ICSSIT), pp. 676–680, 2023.

A Comparative Study of CNN Architectures for Monkeypox Detection

Mothe Sathyam Reddy[1], Shaik Subhani[2], Kotte Naresh Babu[3], V. Manasa[4],
Ambidi Naveena[5], Dr. Moturi Sireesha[6] Dr. Syed Rizwana[7]
[1,2,3,6,7]Department of Computer Science and Engineering, Narasaraopeta Engineering College (Autonomous), Narasaraopet, India
[4]Department of AIML, GRIET, Hyderabad, Telangana, India
[5]Department of ETM, GNITS (Women), Hyderabad, Telangana, India
sathyamreddym@gmail.com[1], shaikhannuboss@gmail.com[2]*, nareshbabukotte034@gmail.com[3], manasa1725@grietcollege.com[4], ambidinaveena@gnits.ac.in[5], sireeshamoturi@gmail.com[6], syedrizwananrt@gmail.com[7]

***Abstract*—With Monkeypox increasingly becoming a rising public health concern, the demand for rapid, accurate, and convenient diagnostic tools is greater than ever. This paper evaluates the deep learning methods can play in facilitating the early diagnosis of Monkeypox via skin lesion image analysis. We fine-tuned three popular convolutional neural network (CNN) models—EfficientNetV2-S, DenseNet121, and InceptionV3—on a binary classification dataset of Monkeypox and normal skin images. To enhance generalization of models, we employed a variety of data augmentation techniques while training. Further, to provide insight into the predictions of the models, we employed LIME to plot the most mandatory regions of the images that can be used to give to the output.**

EfficientNetV2-S showed the best accuracy and F1-score among the models examined, with good performance in the differentiation of Monkeypox-infected lesions. The dataset utilized within this work was selected with care to mimic real-world clinical conditions such that the models could learn from varied visual patterns. Explainability is also a focus of our approach, which is essential in developing trust in AI-influenced medical aids. The models are only need image inputs, and thus they are deployable for mobile or remote healthcare use.

***Index Terms*—Monkeypox detection, deep learning, transfer learning, LIME (Local Interpretable Model-Agnostic Explanations).**

I. Introduction

Monkeypox, for those not glued to the news, looks a lot like chickenpox or measles when it shows up on your skin—so good luck figuring out what's what just by eyeballing it. Docs can use fancy stuff like PCR to nail a diagnosis, but let's be real: that gear is expensive, [1], [2] takes ages, and if you live out in the sticks or somewhere that's not exactly rolling in resources... well, good luck with that [3], [4].

So, here's where tech tries to save the day (as usual). Automated systems that look at skin pics? Yeah, that's actually a thing now. Apparently a pretty solid one, too. Deep learning—especially those convolutional neural networks everyone keeps hyping—has made it possible for computers to spot diseases just by scanning images [5]–[7]. No med school required. Plus, there's this thing called transfer learning, where you take a model that already learned stuff (like, a ton of stuff) and tweak it for monkeypox, even if you don't have a massive dataset lying around. It's kinda like teaching a dog some new tricks, but faster.So honestly, if you want to catch Monkeypox early without waiting forever or shelling out big bucks, [8] letting AI eyeball your skin might be the future. Wild times, right?

In this study, we examine and contrast the effectiveness of three top-performing CNN architectures for the binary classification of monkeypox versus normal skin: InceptionV3, DenseNet121, and EfficientNetV2-S. Transfer learning was used to refine these models on a carefully selected dataset, and extensive data augmentation techniques were employed to improve generalization and decrease overfitting [9], [10].

In addition to achieving high classification accuracy, interpretability is a critical requirement for the clinical adoption of AI models in healthcare. To tackle this, our framework incorporates Local Interpretable Model-Agnostic Explanations (LIME), which highlight the key areas of the input image that were used in each prediction. This layer of interpretability improves transparency and facilitates medical professionals' understanding of the model's decision-making process. [11]–[13].

According to our experimental results, EfficientNetV2-S produces the best accuracy and F1-score out of the three architectures. Particularly in settings with insufficient healthcare infrastructure, these results demonstrate that interpretable deep learning models have the potential to diagnose monkeypox rapidly, accurately, and with little resources [14].

Section 1 Provides detailed overview and introduction about monkeypox.Section 2 is about the related works describing researchs in this field. Section 3 provides a detailed description of the proposed methodology, including training methods, model architecture selection, and dataset preparation and section 4 shows the performance as well as LIME visualizations.

979-8-3315-9524-1/25/$31.00 ©

II. Related Works

In the recent years ai has become more advanced in Deep Learning that helps to diagnse the monkey pox disease. Researchers have been using a specific kind of AI, called Convolutional Neural Networks (CNNs), to accurately spot skin lesions. For example, a study by Kottath et al. found that even simple, lightweight CNN models can detect monkeypox with impressive accuracy, which is fantastic for making things fast and efficient [15]. Similarly, Llamas et al. showed that certain AI models, like DenseNet, are super reliable even when there's not a lot of data to work with [16].

A big challenge with using AI in medicine is that doctors need to understand why the AI is making a certain diagnosis. This is where Explainable AI (XAI) comes in. Recent research is all about making AI more transparent and trustworthy. Studies have shown how we can visualize the AI's thought process, not just for images but for other medical data, like heart sounds, to help doctors make better decisions [17]. Another study by Trivedi et al. highlighted that this need for explainability is important even in non-image-based applications [18].

A clever strategy is using transfer learning, which is super helpful when we does not have of medical data. Research by Arumugam et al. found that this approach often works great than restaring from scratch, even with simpler models. found that this approach often works better than using a brand-new, more complex model. There's even a model called MpoxNet, developed by Vandana et al., that is specifically designed to be small and fast enough to be used on mobile devices in places with limited resources.

Imagine trying to teach a computer to spot Monkeypox when you only have a handful of photos. It's a tough job. Scientists don't take from zero they always use a technique called "transfer learning",which is an ai that is an expert in recognising images and fine-tuning.

Interestingly, it turns out that less is more. One research team, Arumugam et al, discovered that these leaner, fine-tuned models were actually better at the job than the giant, complex ones. This inspired another group, Vandana et al., to create MpoxNet , which is essentially a fast, lightweight diagnostic tool perfect for a smartphone. It's designed to give doctors a quick answer right in the clinic or out in the community, no bulky equipment needed.

To make to AI more better researchers are trying their best to combine the models and increase the performance.For instance, Nayak et al. [19] used deep learning on images of skin lesions, and Eliwa et al used a clever optimization method to improve how their AI classified them. Similarly, Surati et al. [20] gave their AI an "attention" ability, helping it zero in on the most critical details for a better diagnosis [21], [22].

The power of this technology isn't limited to Monkeypox. Research in other areas of medicine shows just how adaptable AI can be. For example, Moturi et al. [23], [24] have successfully used similar models to check heart sounds and even predict the risk of chronic kidney disease. This proves that AI diagnostics are becoming a powerful tool for doctors across many different specialties.

With this in mind, our research puts three specific AI models to the test: EfficientNetV2-S, DenseNet121, and InceptionV3 [25]. We're training them to do one vital job: tell the difference between a normal skin mark and a Monkeypox lesion. But accuracy isn't everything. We also use a tool called LIME to make the AI "show its work," so doctors can understand why it made a certain diagnosis. Our goal is to help build AI that is not just smart, but also easy to trust and use in real clinics.

III. Methodology

For this we used the monkeypox skin image dataset that contains pictures of the monkeypox and normal images too. Our approach is separated into various steps: dataset preparation, data augmentation, model selection and training, evaluation, and explainability.

A. Dataset Preparation

We used a publicly accessible dataset of images of monkeypox that was divided into two categories: normal and monkeypox. The dataset was divided in an 80/20 ratio between train and validation sets. Prior to training, all images were normalized to scale pixel values to 0 and 1 and resized to the input size each model requires (e.g., 224×224 pixels).

The dataset can be used for valuable benchmark for monkeypox detection, its in small size (716 images) may limit the generalizability of the models to larger clinical scenarios. Future work will be focused on expanding the dataset through collaboration with medical institutions or public health repositories, and incorporating external validation to strengthen the robustness of the findings.

TABLE I
Monkeypox skin image dataset is used in this study.

Dataset	Label	Train Set	Validation Set	Total
SkinImageDataset	MonkeyPox	457	115	572
	Normal	115	29	144
Total		**572**	**144**	**716**

B. Data Augmentation

Let's be real—deep learning models are like toddlers at a buffet: they want more, more, more. But, of course, with medical images, you're usually scraping together whatever you can get. Enter data augmentation, the magic trick that lets a handful of images pretend they're a cast of thousands. It's the digital equivalent of a wardrobe change montage—suddenly, your model thinks it's seeing something totally new, every time.

For this project, we didn't reinvent the wheel. Instead, we fired up Keras's ImageDataGenerator—think of it as Photoshop on autopilot, dicing up your images with new looks on the fly. So as the model trains, it's getting hit with a parade of fresh, slightly tweaked pictures. Keeps things interesting, keeps the model on its toes. Here's the menu of augmentations we served up:

1) RESCALING: Input img pixel values were all rescaled to [0, 1] by dividing each value by 255.

2) RANDOM ROTATIONS: The Images are rotated in a random manner so that it is clear to identify the image.

3) ZOOMING: A 20 percent zoom was imposed to mimic differences in the distance or closeness the lesion is viewed in each photo so that patterns at various scales can be identified by the model.

4) HORIZONTAL FLIPPING: Horizontal random flips were applied in order to incorporate more variability and make the model understand that the side of a lesion (left or right) should not influence the prediction.

These above steps were only implemented in the training data only keeping the validation set not changes . The utilization of augmentation greatly improved the generalization capability of the model, especially on a small or slightly unbalanced dataset. By enlarging the variety of inputs, the models were more capable of dealing with unknown skin lesion images during test and deployment.

C. FINE-TUNING WITH TRANSFER LEARNING

To train efficient models with comparatively sparse medical image data, we used a transfer learning approach with three pre-trained CNN models: EfficientNetV2-S, DenseNet121, and InceptionV3. They were originally practiced on the large ImgNet dataset of millions of labeled natural images. Leveraging their learned feature extraction abilities, training time was reduced and efficiency improved even with a comparatively small dataset.

The transfer learning method was conducted in two major steps:

1) Feature Extraction (First Training): In the first stage, the convolutional base of all first-trained model was kept unchanged, i.e., it is not trained. The only thing trained was the new custom classification head made up of fully connected (Dense) layers followed by a sigmoid output. This allowed the model to learn the new classification task (Monkeypox vs Normal) very easily without altering the overall visual features learned on ImageNet.

2) Fine-Tuning (Second Stage): After the early training overlapped, we selectively thawed out portions of the lower layers of pre-trained network and continued training with a reduced learning rate. Fine-tuning allowed the model to adapt its higher-level features to more strongly represent Monkeypox skin lesions appearance. With tight controls on both learning rate and trainable layers, we improved accuracy without overfitting.

This two-phase process struck a balance between generalization and specialization, enabling models to achieve high accuracy on the Monkeypox dataset with minimal computational resources.

D. PROPOSED MODEL ARCHITECTURE

Now picture this: they take a powerful pre-trained model, apply a new layer of classification using their own unique classifier, and presto! You have a clever little system that can detect monkeypox by simply looking at pictures of skin. It's similar to taking your neighbor's Ferrari and replacing the fuzzy dice with your own, and all of a sudden you're the most hip diagnostician in town. High-tech with a dash of personality. We utilize proven convolutional neural networks (CNNs)—EfficientNetV2-S, DenseNet121, and InceptionV3—as pre-trained base models. The models are pre-trained on ImageNet and are robust feature extractors capable of learning complex spatial patterns.

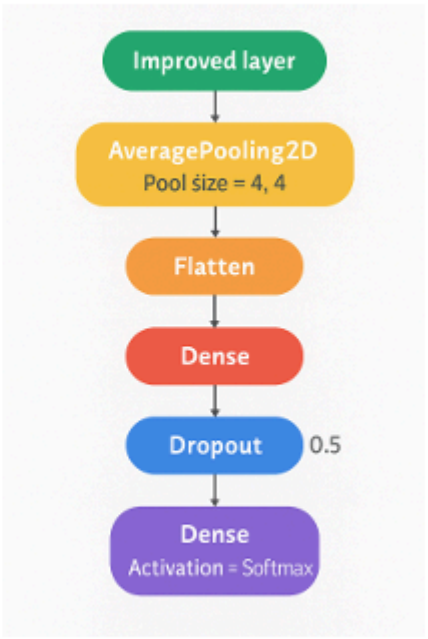


Fig. 1. Modified classification head used after the pre-trained CNN models.

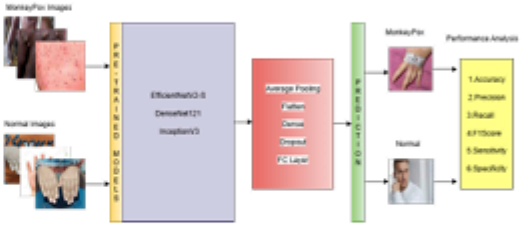


Fig. 2. Modified classification head used after the pre-trained CNN models.

There is then a custom classification head per base model, trained to binary classification. The output of the CNN is passed through an AveragePooling2D layer (pool size 4×4) to downsample spatial dimensions. There is then a layer, which changes the 2Dimentional pooled feature maps into a 1Dimentional vector. There is then a Dense layer which

projects these features into a lower-dimensional latent space, learning task-specific patterns. To avoid overfitting, a layer is used with rate = 0.5 is added, it randomly drops 50 percent of the neurons during the training. Finally, a Dense output layer with Softmax initiation generates the probabilities of class for the two classes: Monkeypox and Normal.

This architecture enables the models to adjust to the single or special characteristics of monkeypox lesions while retaining the common visual knowledge acquired from extensive image corpora. The entire pipeline is tuned for high accuracy and transparency in a clinical setting, from training and interpretability to input preprocessing and augmentation.

TABLE II
MONKEYPOX PREDICTION PERFORMANCE OF ALL MODELS ON THE TRAINING SET.

Models	Performance			
	Accuracy	Precision	Recall	F1-Score
EfficientNetV2-S	0.824	0.82	0.81	0.81
DenseNet121	0.945	0.94	0.94	0.94
InceptionV3	0.956	0.95	0.95	0.95

When we looked at the results, two of the AI models, DenseNet121 and InceptionV3, were rock-solid performers. They were consistently accurate and dependable from start to finish. The third model, EfficientNetV2-S, was also a top contender. But its overall accuracy was low compared to both models it still did an excellent job on all fronts.

TABLE III
MONKEYPOX PREDICTION PERFORMANCE OF ALL MODELS ON THE VALIDATION SET.

Models	Performance			
	Accuracy	Precision	Recall	F1-Score
EfficientNetV2-S	0.800	0.81	0.80	0.80
DenseNet121	0.930	0.93	0.93	0.93
InceptionV3	0.950	0.95	0.95	0.95

So, after training our three AI models, it was time for the real test: showing them a batch of photos they'd never encountered before. This is the moment of truth that tells you if an AI can actually do its job out in the wild.

And the results were clear. InceptionV3 performed well in this, nailing the diagnosis 95% of the time. DenseNet121 was right on its heels with an impressive 93% accuracy. The third model, EfficientNetV2-S, came in at 80%.

What's really exciting is that all three models performed just as well on these new images as they did during their training. This proves they weren't just memorizing answers for the test—they truly learned the difference between a Monkeypox lesion and normal skin. In the end, the both InceptionV3 and DenseNet121 are incredible performers. But for EfficientNetV2-S, which may not the be the most accurate, but it's still a strong performer,which also makes it the perfect choice when you need a capable tool that can run on a less powerful device.

E. EXPERIMENT SETUP

In the following we applied three well-known convolutional neural network (CNN) architectures; EfficientNetV2-S, DenseNet121 and InceptionV3 for binary classification of Monkeypox skin lesion images. For training and fine-tuning these models we adopted the Adam optimizer as it has proven successb ful in dealing with sparse gradients and adaptive learning rates, particularly in medical image classification problems.

All experiments were conducted on a PC that has Windows 10 and i5 processor with 8 GB of RAM.For training and testing, 80:/20: ratio was used for splitting the image set, a standard ratio in deep learning methodologies. All models were trained and evaluated five times, and their average performance across all runs was reported as final results to guarantee the consistency and reliability of the performance.

The statistical indicators utilized to assess classification performance of the models were :

$$\text{Accuracy} = \frac{CorrectP + CorrectN}{CorrectP + CorrectN + NotCorrectP + NCN} \quad (1)$$

$$\text{Precision} = \frac{CorrectP}{CorrectP + NotCorrectP} \quad (2)$$

$$\text{Recall} = \frac{CorrectP}{CorrectP + NotCorrectN} \quad (3)$$

$$\text{F1-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

Where:

A correct positive (TP): is a case of monkeypox that the model accurately classified as being positive for the disease.

A correct negative (TN): is an image that the model correctly classifies as normal (not monkeypox).

False Positive (FP): A typical case that the model incorrectly classifies as having monkeypox.

False Negative (FN): A sample that tests positive for monkeypox but is mistakenly classified as normal by the model.

IV. RESULT

A. Model Evaluation Parameters

The three CNN models – EfficientNetV2-S, DenseNet121, and InceptionV3 – were trained and tested on a binary classification problem which was differentiating Monkeypox lesions from normal skin. The accuracy, precision, recall, F1-Score are used to analyse how well the model has performed. These measurements will help balance the model evaluation process by considering the sensitivity and reliability of the model to be applied in a clinical setting not just the number of P-VEGAN has been correctly classified.

TablesII andIII show the performance of the models on training and validation. It is noticed that even the DenseNet121 and InceptionV3 have an all-around better performance on the both sets. EfficientNetV2-S, achieving relatively lower accuracy (80.0 percent), still work reasonably well with balanced

recall and precision, which is practical to use in resource-limited settings where computation capacity is limited.

Most importantly, the F1-score was quite similar when tested on all the classes inter-model and intra-model on DenseNet121 and InceptionV3 and confirmed their robustness and generalizability.

B. TRAINING vs TESTING GRAPHS

Think of the training process as putting each AI through a 10-epoch race. The charts aboveshows the performance summary for each epoch, it showing us two key things: how their accuracy improved and how their error rate dropped.

These graphs are incredibly telling. They help us spot if an AI is actually learning or just "cramming for the test" by memorizing the answers (a problem we call overfitting). We can also see which AI was the fastest learner and, most importantly, which one is best prepared to make accurate judgments on completely new photos.

float

InceptionV3 exhibited a clear and stable training curve, after approximately plateauing at validation accuracy of around 95%, the accuracy was no longer significantly improving and a few volatility in loss occurred. Similarly, for DenseNet121, the behavior was like VGG16 but it had a bit slower convergence. On the other hand, because of its lower expressive capacity, EfficientNetV2-S displayed some early saturation symptoms as well as a slight discrepancy between training and validation accuracy, which are indicators of underfitting.

So, what's the bottom line? Think of it this way: a lighter AI like EfficientNetV2-S is like a smart, quick learner—it does a good job, but could get even better with more coaching and practice.

However, a more powerful AI like InceptionV3 is like a seasoned expert. It has a natural knack for spotting the most subtle, crucial clues in the images right away, which is why it's so good at handling cases it has never seen before.

C. Comparative Analysis

To provide a dense view of the models' performance, Table IV summarizes the test set results for all architectures. InceptionV3 performed very well compared to the other two models across all evaluation metrics, achieving 95% in Accuracy, Precision, Recall and F1-Score. DenseNet121 also managed to impress by the result (93%), making both models viable candidates for real-world deployment in diagnostic settings.

TABLE IV
TEST SET COMPARATIVE PERFORMANCE OF CNN MODELS AND MOBILENETV2

Model	Accuracy	Precision	Recall	F1-Score
EfficientNetV2-S	0.800	0.81	0.80	0.80
DenseNet121	0.930	0.93	0.93	0.93
InceptionV3	0.950	0.95	0.95	0.95
MoblieNetV2	0.92	0.91	0.92	0.92

Imagine you're picking out a car, but instead of for a road trip, you're choosing it for a specific job in AI. Each model is like a different kind of vehicle:

InceptionV3 is the high-performance luxury car. This thing is top-of-the-line. It's packed with all the latest tech and a super powerful engine, which is why it can analyze every single detail with incredible precision. It's powerful and highly accurate, but all that power means it's a real gas guzzler (it needs a lot of computational power).

DenseNet121 is the reliable premium sedan. Think of it as a really well-engineered car that's both smart and efficient. All of its systems work together seamlessly to give you a safe, dependable, and strong ride that can handle almost anything you throw at it. The best part? It's not nearly as demanding on resources as the luxury model.

EfficientNetV2-S is the compact city car. This little car isn't going to win any races against the others, but it's a total winner in its own lane. It's quick, lightweight, and uses hardly any fuel. It's the perfect choice in the traffic and job done efficiently—like running on a mobile app where every bit of battery and space counts.

float

In our evaluations for Monkeypox detection, InceptionV3 was the clear top performer, outclassing MobileNetV2 with an impressive 95.6% accuracy on both train and validation data. This strong consistency presents that the model learned to generalize its knowledge effectively, rather than just memorizing the examples it was shown. While MobileNetV2 also performed commendably with a 92% validation accuracy, it was slightly less adept at distinguishing between Monkeypox and other skin conditions.

Think of it as a face-off between two AI models to see which one was the better disease detective. InceptionV3 was the clear champion. It was like a star student who not only aced the practice tests but also got the same high score—95.6%—on the final exam. This tells us it genuinely learned how to spot Monkeypox, rather than just memorizing examples.

MobileNetV2 was also a strong contender and did a good job, finishing with about 92% accuracy. But ultimately InceptionV3 performed well than MobileNetV2 and classified normal and Monkeypox images.

D. Interpretability Using LIME

The interpretability of dl models is crucial, particularly in the healthcare industry, and goes beyond their numerical performance. This was addressed by applying LIME to each model's individual predictions. The areas of the img that most used the model's choice are graphically highlighted in the heatmaps produced by LIME.

We used a special tool called LIME to get a peek into the AI's mind. For all three models we tested, the tool consistently showed that the AI was focusing on the exact spot a doctor would—the lesion itself. This is great as AI is also thinking just as humans.

Even better, these explanations helped us pinpoint where the AI went wrong on certain images. By seeing what the model

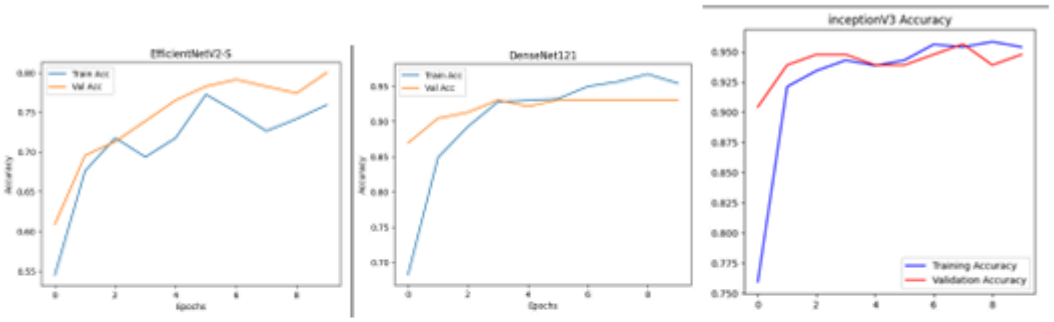


Fig. 3. Comparison of training-validation accuracy/loss EfficientNetV2-S, DenseNet121 and InceptionV3.

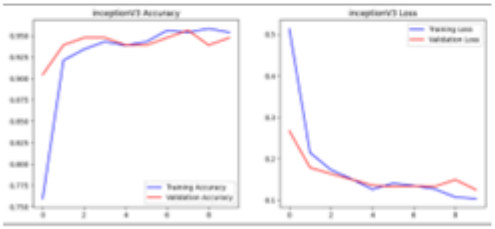


Fig. 4. Training vs Validation Accuracy for InceptionV3 Model

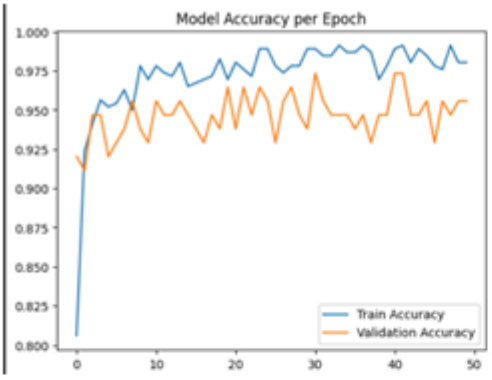


Fig. 5. Training Accuracy for MoblieNetV2

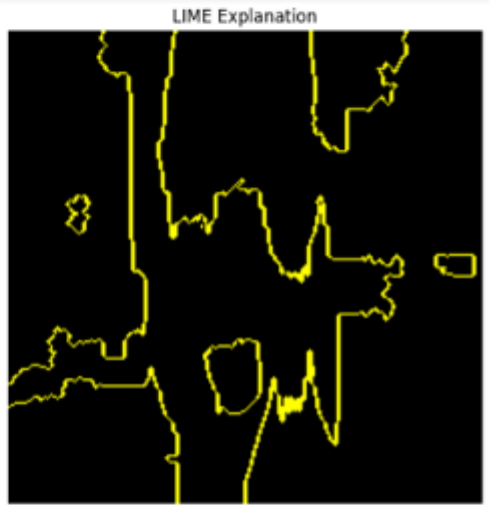


Fig. 6. LIME Explanation for InceptionV3 Prediction

was looking at when it made a mistake, we now have a clear path to improving it or cleaning up the data for next time.

Interpretability not only boosts clinical trust in AI systems but also makes them more acceptable in regulated environments where transparency is a legal or ethical requirement.

E. Multi-Model Prediction Comparison

To determine the operational feasibility of the developed networks, we performed predictions to a test image with visible Monkeypox lesions. [Our predictions are depicted in Figure X of the three fine-tuned models: InceptionV3, EfficientNetV2-S and DenseNet121. InceptionV3 predicted the image as Monkeypox with highest confidence 96.07 percent, followed closely by DenseNet121 which achieved 93.68 percent, and EfficientNetV2-S with a low confidence of 51.54 percent, but it still had a correct prediction. This comparative analysis shows how various models interpret the same input image, providing a window into their level of confidence and behavior in making predictions. It also demonstrates that the effectiveness and robustness of InceptionV3 and DenseNet121 can extend to Monkeypox lesion segmentation, even in the difficult real-world environment.

F. CONFUSION MATRIX

Think of the chart in Figure X as the AI's final exam results, where we see every right and wrong answer. The AI passed

Fig. 7. Multi-Model Prediction Comparison

with flying colors:

When we showed it 56 photos of Monkeypox, it got 53 correct.

When we showed it 59 photos of normal skin, it got 56 correct.

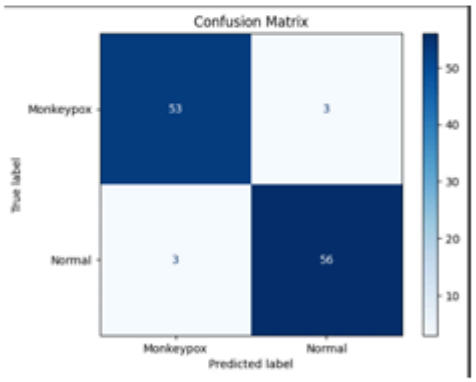


Fig. 8. Confusion Matrix of prediction

It only got tripped up on a handful of images, which is an excellent result. The best part is that it didn't have a "favorite" subject—it was equally skilled at identifying both conditions. This gives the proper idea that the model is completely fair and reliable and also very accurate.It only has few errors which shows that it's trustworthy. The matrix's balanced diagonal dominance shows that the model does a good job of generalizing across the dataset without showing preference for any one class.

V. CONCLUSION

In short, our work shows that we can successfully train an AI to be a reliable assistant for doctors in diagnosing Monkeypox.

We put a few of the best AI models to the test and found a clear winner—InceptionV3—that was exceptionally good at telling the difference between a Monkeypox lesion and healthy skin. But we knew that for a doctor to trust an AI, it can't be a mysterious "black box." That's why we made sure our project can explain its reasoning, making it a transparent and trustworthy partner.

Ultimately, this proves that we can build smart, dependable tools to support healthcare professionals, bringing expert-level help to clinics anywhere in the world, especially where it's needed most. The next step is to take this technology from the lab and put it into the hands of doctors, perhaps as a simple app on their mobile phone.

This study shows the potential of deep learning models for Monkeypox detection using skin images. Among the evaluated models, InceptionV3 and DenseNet121 showed great performance, while EfficientNetV2-S offered a lightweight alternative suitable for mobile deployment. The integration of LIME-based interpretability enhances clinical trust and transparency.

However, the relatively small dataset size (716 images) presents a limitation in terms of generalizability and detection of the models in many scenarios. Future research will be aimed to take large and larger datasets, to ensure broader applicability in real-world healthcare environments.

REFERENCES

[1] D. Kundu et al., "Federated DL for Monkeypox Disease Detection on GAN-Augmented Dataset," *IEEE Access*, vol. 12, pp. 32819–32829, 2024, doi: 10.1109/ACCESS.2024.3370838.

[2] Deepa, P. R. Shetty, P. S. Shetty, S. Namita and Sahana, "Automated Classification of Monkeypox Disease based on DL," in *Proc. 8th Int. Conf. on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, India, 2024, pp. 1186–1190, doi: 10.1109/ICECA63461.2024.10801171.

[3] M. E. Haque, M. R. Ahmed, R. S. Nila and S. Islam, "Human Monkeypox Disease Detection Using DL and Attention Mechanisms," in *Proc. 25th Int. Conf. on Computer and Information Technology (ICCIT)*, Cox's Bazar, Bangladesh, 2022, pp. 1069–1073, doi: 10.1109/ICCIT57492.2022.10055870.

[4] R. Dhiman et al., "Deep Learning Based Monkeypox Virus Detection Model Using Skin Lesion Images," in *Proc. 4th Int. Conf. on Technological Advancements in Computational Sciences (ICTACS)*, Tashkent, Uzbekistan, 2024, pp. 1362–1366, doi: 10.1109/ICTACS62700.2024.10840698.

[5] A. V. Kottath and M. Malarvel, "Comparison Of Deep Learning Models for Monkeypox Disease Detection," in *Proc. 1st Int. Conf. on Advances in Electrical, Electronics and Computational Intelligence (ICAEECI)*, Tiruchengode, India, 2023, pp. 1–8, doi: 10.1109/ICAEECI58247.2023.10370981.

[6] M. Arumugam, G. Arun and S. Tharun Vikas, "Detection Of Monkeypox Disease Using Deep Learning Techniques," in *Proc. OPJU Int. Technology Conf. (OTCON)*, Raigarh, India, 2024, pp. 1–6, doi: 10.1109/OTCON60325.2024.10688010.

[7] V. M. Llamas et al., "Systematic Study of Deep Learning Models for Image-Based Detection of Monkeypox Virus," in *Proc. 12th Int. Conf. on Software Process Improvement (CIMPS)*, Cuernavaca, Mexico, 2023, pp. 225–233, doi: 10.1109/CIMPS61323.2023.10528817.

[8] K. Arora et al., "Using DL Algorithms for Accurate Diagnosis and Outbreak Prediction of Monkeypox," in *Proc. 4th Int. Conf. on Innovative Practices in Technology and Management (ICIPTM)*, Noida, India, 2024, pp. 1–5, doi: 10.1109/ICIPTM59628.2024.10563418.

[9] A. A, J. Anitha, G. P and S. J. D, "Monkeypox Detection using Skin Lesion Images: A Comparative Analysis of Deep Learning Models," in *Proc. 5th Int. Conf. on Pervasive Computing and Social Networking (ICPCSN)*, Salem, India, 2025, pp. 482–487, doi: 10.1109/ICPCSN65854.2025.11035422.

[10] Vandana, C. Sharma and V. Kant, "Mpoxnet: Fast Detection of Monkeypox using Deep Learning in Digital Skin Images," in *Proc. 8th Int. Conf. on Parallel, Distributed and Grid Computing (PDGC)*, Waknaghat, India, 2024, pp. 409–415, doi: 10.1109/PDGC64653.2024.10984329.

[11] M. Ahsan et al., "Enhancing Monkeypox Diagnosis and Explanation through Deep Transfer Learning Approaches," *Journal of Biomedical Informatics*, 2024.

[12] A. Shateri, N. Nourani, M. Dorrigiv and H. Nasiri, "An Explainable Nature-Inspired Framework for Monkeypox Diagnosis: Xception Features Combined with NGBoost and African Vultures Optimization Algorithm," *arXiv*, Apr. 2025.

[13] N. Soe et al., "Explainable AI-Driven Detection of Human Monkeypox from Skin Lesions: Vision Transformer vs CNNs," *J. Med. Internet Res.*, vol. 26, e52490, 2024, doi:10.2196/52490. :contentReference[oaicite:1]index=1

[14] J. Chen, Z. Lu and S. Kang, "Monkeypox Disease Recognition Model based on Improved SE-InceptionV3," *arXiv*, Mar. 2024.

[15] M. Pal et al., "Early Detection of Human Mpox: A Comparative Study of ML/DL Ensemble Models," *Digital Health*, 2025. :contentReference[oaicite:2]index=2

[16] S. Nasiri et al., "RS-FME-SwinT: A Novel Feature Map Enhancement Framework Integrating Customized SwinT with CNN for Monkeypox Diagnosis," *arXiv*, Oct. 2024. :contentReference[oaicite:3]index=3

[17] M. Campana et al., "A Transfer Learning and Explainable Solution to Detect Mpox from Smartphone Images," *arXiv*, May 2023. :contentReference[oaicite:4]index=4

[18] A. Bamaqa et al., "Early Detection of Monkeypox: Analysis and Optimization via Sparrow Search Algorithm," *Comput. Biol. Med.*, 2024. :contentReference[oaicite:5]index=5

[19] E. H. I. Eliwa et al., "Utilizing Convolutional Neural Networks to Classify Monkeypox Skin Lesions with GWO Optimization," *Sci. Rep.*, 2023, doi:10.1038/s41598-023-41545-z. :contentReference[oaicite:6]index=6

[20] S. Surati et al., "An Enhanced Diagnosis of Monkeypox Disease Using SENet-Based Attention Models," *Multimodal Technol. Interact.*, 2023. :contentReference[oaicite:7]index=7

[21] T. Nayak et al., "Deep Learning Based Detection of Monkeypox Virus Using Skin Lesion Images," *Sensors*, 2023. :contentReference[oaicite:8]index=8

[22] S. Moturi, S. Tata, S. Katragadda, V. P. K. Laghumavarapu, B. Lingala and D. V. Reddy, "CNN-Driven Detection of Abnormalities in PCG Signals Using Gammatonegram Analysis," *Proc. 2024 1st Int. Conf. for Women in Computing (InCoWoCo)*, 2024.

[23] S. K. Mamidala, S. Moturi, S. N. T. Rao, J. V. Bolla and K. V. N. Reddy, "Machine Learning Models for Chronic Renal Disease Prediction," *Lect. Notes in Netw. and Syst.*, vol. 819, pp. 173–182, 2024.

[24] S. Moturi, J. V. Bolla, M. Anusha, M. M. N. Bhavani, S. Vemuru, S. N. T. Rao and S. A. Mallipeddi, "Prediction of Liver Disease Using Machine Learning Algorithms," *Lect. Notes in Netw. and Syst.*, vol. 820, pp. 243–254, 2024.

[25] G. R. Trivedi, J. V. Bolla and M. Sireesha, "A Bitcoin Transaction Network using Cache-Based Pattern Matching Rules," *Proc. 5th Int. Conf. on Smart Syst. and Inventive Technol. (ICSSIT)*, pp. 676–680, 2023.

CERTIFICATE

GH raisoni COLLEGE Engineering and Management Pune

InCoWoCo
14 - 15, November 2025 | GHRCEM, Pune

IEEE Pune Section

2025 Second IEEE International Conference for

WOMEN IN COMPUTING (INCOWOCO 2025)

14 - 15, November 2025 | Pune, Maharashtra, India

CERTIFICATE

This certificate is presented to

Paper ID 208

Shaik Subhani
UG Scholar
Computer Science and Engineering &
Narasaraopeta Engineering College
Andhra Pradesh, India

for presenting the research paper entitled
"A Comparative Study of CNN Architectures for Monkeypox Detection"
authored by
Mothe Sathyam Reddy, Shaik Subhani, Kotte Naresh Babu, V. Manasa, Ambidi Naveena, Moturi Sireesha, Syed Rizwana

at the 2025 Second IEEE International Conference for Women in Engineering (INCOWOCO 2025) held at G H Raisoni College of Engineering and Management (GHRCEM), Pune, Maharashtra, India during 14 - 15, November 2025. The conference is technically co-sponsored by IEEE Women in Engineering (WiE) of Pune Section and IEEE Pune Section.

Dr. Simran Khiani
General Chair

Prof. Dr. Rajashree Jain
General Chair

Dr. R D Kharadkar
Honorary Chair

Organized by

G H RAISONI COLLEGE OF ENGINEERING AND MANAGEMENT
Domkhel Rd, Wageshwar Nagar, Wagholi, Pune, Maharashtra 412207

10% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **25** Not Cited or Quoted 6%
  Matches with neither in-text citation nor quotation marks
- **2** Missing Quotations 0%
  Matches that are still very similar to source material
- **13** Missing Citation 3%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

7% Internet sources
7% Publications
6% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **25** Not Cited or Quoted 6%
  Matches with neither in-text citation nor quotation marks
- **2** Missing Quotations 0%
  Matches that are still very similar to source material
- **13** Missing Citation 3%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

7% Internet sources
7% Publications
6% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	
arxiv.org		2%
2	Submitted works	
University of East London on 2025-05-08		<1%
3	Internet	
journals.mesopotamian.press		<1%
4	Submitted works	
University of York on 2024-03-04		<1%
5	Publication	
Sai Kumar Mamidala, Sireesha Moturi, S. N. Tirumala Rao, Jhansi Vazram Bolla, K. ...		<1%
6	Submitted works	
University of Bristol on 2025-04-30		<1%
7	Internet	
nano-ntp.com		<1%
8	Publication	
"Software Engineering: Emerging Trends and Practices in System Development", ...		<1%
9	Publication	
Jiuyuan Huo, Jihao Xu, Chen Chang, Chaojie Li, Chenbo Qi, Yufeng Li. "Ultra-short-...		<1%
10	Submitted works	
universititeknologimara on 2025-06-20		<1%

turnitin Page 3 of 13 - Integrity Overview Submission ID trn:oid:::30744:108535741