AUTOMATED ADVANCED ACNE LESION ANALYSIS AND MONITORING WITH ARTIFICIAL INTELLIGENCE.

24_25J_210

Final Report

Muhandiram Ralalage Maleesha Gimmana - IT21176456

BSc (Hons) in Information Technology Specializing in Data Science

Department of Information Technology

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

AUTOMATED ADVANCED ACNE LESION ANALYSIS AND MONITORING WITH ARTIFICIAL INTELLIGENCE.

Muhandiram Ralalage Maleesha Gimmana

IT21176456

Bachelor of Science (Hons) in Information Technology Specializing in Data Science

Department of Information Technology

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

DECLARATION

I declare that this is my own work and this dissertation1 does not incorporate without acknowledgment any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books)

Signature:

Date: 2025/04/10

The above candidate has carried out research for the bachelor's degree Dissertation under my supervision.

Signature of the supervisor:

Date: 2025/04/11

(Mr. Samadhi Rathnayake)

Signature of the co-supervisor: Date:

(Dr. Junius Anjana)

ABSTRACT

Acne is a prevalent chronic inflammatory skin condition affecting millions worldwide, with significant physical and psychological impacts. Current diagnostic and severity assessment methods rely heavily on subjective clinical evaluation, leading to inconsistencies in treatment. To address this challenge, this study explores the integration of artificial intelligence (AI) and machine learning (ML) for automated acne detection and severity classification. Leveraging advanced deep learning architectures such as Convolutional Neural Networks (CNNs) and data augmentation techniques, the proposed system analyzes dermatological images to accurately identify lesion subtypes and grade severity levels. The framework aims to reduce diagnostic variability, enhance efficiency, and support personalized treatment plans. Preliminary results demonstrate high accuracy in lesion segmentation and classification, validated against dermatologist-annotated datasets. By combining explainable AI (XAI) for interpretability and scalable deployment, this research offers a promising tool for improving acne management in clinical and remote settings. The study underscores the potential of AI-driven solutions to standardize acne diagnosis, optimize therapeutic outcomes, and expand access to dermatological care.

Keywords: Acne vulgaris, skin inflammation, chronic disease, machine learning, personalized care.

ACKNOWLEDGMENT

I wish to express my profound gratitude to all the individuals who have supported me in the continuation of my fourth-year research project. Firstly, I extend my sincere thanks to my Research Project Supervisor, Mr. Samadhi Rathnayake, for his invaluable guidance, encouragement, and support. Additionally, I am deeply appreciative of the advice and assistance provided by our Co-Supervisor, Dr. Junius Anjana.

I would also like to thank the CDAP panel for their insightful guidance throughout the duration of our project. Furthermore, my sincere thanks go to our external supervisors, Dr. Saman Gunasekara and Dr. Kasun Nayanakantha, for their coordination and provision of crucial information, which significantly aided the initiation of our research.

Moreover, I am grateful for the contributions made by our Research Project Leader, Gimmana M.R.M., and fellow team member, Perera W.A.S.K., whose support was invaluable throughout this project. Finally, I would like to extend my heartfelt gratitude to my parents and friends for their unwavering support, which was instrumental in the success of this research project.

CONTENT

DECLARATION i

ABSTRACT ii

ACKNOWLEDGMENT iii

LIST OF FIGURES vi

LIST OF TABLES viii

LIST OF ABBREVIATION viii

1 Introduction 1

1.1 Background Literature 1

1.2 Literature Survey 5

1.3 Research Gap 8

1.4 Research Problem 10

1.5 Research Objectives 11

1.5.1 Main Objectives 11

1.5.2 Sub Objectives 12

1.6 Requirement Gathering and Feasibility Study 14

1.6.1 Requirement Gathering and Analysis 14

1.6.2 Feasibility Study 19

1.7 System Design 22

1.7.1 Overall System Design 23

1.7.2 Acne Detection and Severity Assessment Component's Architecture Diagram 24
1.8 Key Pillars of the Research Component 25
1.8.1 Acne Detection Model 25
1.8.2 Lesion Localization with OpenCV 25
1.8.3 Acne Severity Assessment 26
2 Methodology 26
2.1 Methodology – Acne Detection 26
2.1.1 Data Collection 26
2.1.2 Data Preprocessing 28
2.1.3 Model Building and Training 32
2.2 Methodology – Acne Severity Assesment 52
2.2.1 Data Collection 52
2.2.2 Data Preprocessing 55
2.2.3 Model Building and Training 58
2.3 Commercialization Aspect of the Product 83
2.4 Testing and Implementation 85
2.4.1 System Implementation 85
2.4.2 Testing and Evaluation 85
2.4.3 Real-World Validation 86
2.4.4 Backend Implementation 86
2.4.5 Frontend Implementation 95
3 Results and Discussions 100
3.1 Results 100

3.2 Research Findings 100
3.2.1 Acne Detection 100
3.2.2 Acne Severity Detection 111
3.3 Discussion 118
4 Conclusion 123
5 References 125
6 Glossary 127
7 Appendices 129

LIST OF FIGURES

Image 1 : DermNet Website for Data Collection 27
Image 2 : DermNet Website for Data Collection 28
Image 3 : Proof of Data Valdation with External Supervisors 28
Image 4 : Color based Augmentation 31
Image 5 : Data Augmentation using ImageDataGenerator 31
Image 6 : Acne Lesion Laclization using OpenCV 57
Image 7 : Model Loading 87
Image 8 : CORS of Flask App 87
Image 9 : Detection Endpoints 88
Image 10 : Detection routs handling 91
Image 11 : Database Integration 94
Image 12 : Testing using PostMan 95
Image 13 : Frontend Implmentation 96

Image 14 : Doctor Dashboard 97
Image 15 : Acne Detection and Severity Assessment 98
Image 16 : Patients' Results..... 98
Image 17 : Healing Progress Dashboard 99
Image 18 :Confution Matrix of CNN 102
Image 19 : Accuracy and loss graphs of CNN 102
Image 20 : Confusion Matrix of VGG16 103
Image 21: Accuracy and Loss graphs of VGG16..... 104
Image 22 : Confusion Matrix of VGG19 105
Image 23:Accuracy and Loss graphs of VGG19 105
Image 24:Confusion Matrix ResNet 107
Image 25:Accuracy and Loss graphs of ResNet 107
Image 26:Confusion Matrix of EfficientNet 109
Image 27: Accuracy and Loss graphs of EfficientNet..... 109
Image 28:Confusion Matrix of Siamese neural Networks 110
Image 29: Accuracy and Loss graphs of Siamese Neural Networks.....111
Image 30: Accuracy and Loss graphs of EfficientNet.....112
Image 31: Confusion Matrix of CNN 114
Image 32:Accuracy and Loss graphs of CNN.....114
Image 33:Confusion Matrix of VGG16 116
Image 34:Accuracy and Loss graphs of VGG16 116
Image 35 : Image 3 18 - Confusion Matrix of ViT 118
Image 36:Accuracy and Loss graphs of ViT 118
Image 37 :Logo 129
Image 38 : Turnitin Report 131

LIST OF TABLES

Table 1 : Overview of existing methods and technologies for Acne Lesion Detection and severity analysis 8
Table 2 : Comparison of the features addressed by the previous research studies and the features that will be covered in the proposed solution 10
Table 3 :Budget Allocation..... 22
Table 4:Accuacy and Error Rate table of Acne Detection Models..... 101
Table 5 : Accuracy and Loass Rates of Acne Severity Detection Models 112

LIST OF ABBREVIATION

Abbreviation	Long Form
CNN	Convolutional Nerual Network
UI	User Interface
DB	Database
ML	Machinese Learning
XAI	Explainable Artificial Intelligence
RGB	Red , Green , Blue
AI	Artificial Intelligence

1 Introduction

1.1 Background Literature

Acne is a chronic inflammatory disease of the pilosebaceous unit. The blockage leads to inflammation, and acne is a very common problem that affects people of all ages especially during teen years into young adulthood. [1] And mostly acne appears on face, but it can also appear to the chest, shoulders and back. These are the areas of our skin where we secrete a lot of oil: Hence on those lines things that can lead to acne coming up is,

- Hormonal changes (e.g.: during puberty, menstrual cycles, pregnancy)
- Excess oil production
- Buildup of dead skin cells
- Bacterial infection
- Certain medications
- Genetics
- Stress

Acne Severity and Impact

Most people have probably suffered from acne at some point in their lives and it can range in severity. The different stages of acne are important so that you can adequately address the issue at hand. We take a deep dive into these stages to have an in-depth look at this skin care issue. [2]

- Stage 1: Mild Acne

Mild acne typically consists of a few comedones (blackheads and whiteheads) on the face with occasional inflamed pustules, which are visible on the skin as red bumps that contain pus. The accompanying blemishes are usually very small and not inflamed. This stage can

usually be treated with over the counter (OTC) acne products, which typically come in the form of cleansers, topical creams or gels containing benzoyl peroxide and salicylic acid.

- Stage 2: Moderate Acne

Advanced disease has increasing number of comedones but also papules (small red elevated lesions) and pustules. The lesions can alternatively develop in size and are clinically more inflamed. At this point, OTC treatments might not be enough and you may need prescription medications consisting of topical retinoids or oral antibiotics to help control acne.

- Stage 3: Severe Acne

In the Severe stage, It is accompanied by a sizable upsurge in inflamed lesions and large papules, pustule (pimples), nodular(cyst) over many body parts or of extensive appearance. There is also quite a few comedones remaining. Scaring gets started at this point in time, so much better to contact medical attention. Most often, a combination of topical retinoids and oral antibiotics as well hormonal therapy can effectively treat the condition under prescription level care.

- Stage 4: High Severe Acne

High Severe acne, the most extreme stage, is described as a large number of inflamed lesions with numerous papules and pustulating cystic (big lumps full off pus) lesions. As the condition is so common and scarring on a large scale physically/ emotionally we found this stage one of the most difficult via which to deliver care. Severe acne is treated with aggressive therapy, which often requires a combination of prescription oral retinoids, antibiotics and hormonal therapies to improve the disease while reducing long-term scarring.

Figure 1-1 - Severity Levels of Acne

Importance of Acne Staging:

- Appropriate Treatment Selection:

The best treatment depends on the stage of acne and can be addressed with OTC products for mild cases, to prescription medications or professional therapies for moderate- severe stages. Determining the stage properly allows healthcare providers to provide appropriate interventions, making it more likely for successful management of disease.

- Optimizing Treatment Effectiveness:

Also, various stages of acne may need a specific treatment strategy like topical retinoids or oral antibiotics or hormonal therapy. Precise staging of acne is essential to target the most tailored combination treatments and increase the odds for reaching a desired clinical response.

- Preventing Progression and Complications:

Acne can cause permanent, in- flamed and cystic lesions so´ it is important to detect the disease early on and treat adequately. Such debilitating complications could be prevented by the timely intervention that can only come with proper staging.

- Managing Patient Expectations:

Proper acne staging informs the patient what to expect in terms of severity and duration, helping set realistic expectations and potential adherence with treatment planned.

- Monitoring Disease Progress:

Accurate staging of acne at the time of initial consultation and subsequent follow-up visits, enables health care providers to monitor changes in severity or reduction over a period, thereby guiding modification in treatment regimens. And compare responses to interventions with an objective baseline.

It seems logical that optimal therapeutic results need to know the right stages of acne and will significantly help in tailoring management plans accordingly It also helps you prevent disease progress and hence put reasonable expectations. Prescribing appropriate treatments mutually decided with patients is so satisfying instead of wasted trials on ineffective ones. With the power of precise acne staging, healthcare providers can address long-term skin well-being and increase quality of life for hundreds of thousands who are afflicted by this common condition.

Acne has been diagnosed, severity assessment and wound healing progress tracking in different studies using Machine Learning. For example, the studies have proved how accurately acne lesions can be detected and its severity level diagnosis that could generate from merging image processing techniques. These studies indicate the potential to integrate machine learning applications within routine dermatological practices could assist in enhancing acne diagnosis accuracy.

"DermaScope AI" is a novel application that uses cutting-edge deep learning technology to enhance acne diagnosis and evaluation. The app can precisely detect several stages of acne. DermaScope AI is a cutting-edge alternative to conventional diagnosis techniques, making it quicker and simpler for medical professionals to diagnose acne, particularly in places with limited access to specialists.

1.2 Literature Survey

Rohit Yadav, Aashika Jain, and Sanjiv Sharma proposed a deep learning-based acne detection system utilizing a ResNet-18 architecture. Their model was trained on a dataset encompassing diverse acne severities and incorporated data augmentation for enhanced robustness. The system categorized acne into four levels (normal, mild, moderate, severe) and achieved a reported accuracy of 90%. While their approach demonstrates promising results, the reliance on a specific dataset limits the model's generalizability to individuals with varying skin tones. Furthermore, the absence of detailed information regarding acne localization and quantification hinders a comprehensive assessment of the system's capabilities. Future research should focus on expanding the dataset to include diverse skin types, refining acne localization techniques, and exploring the correlation between acne severity scores and clinical assessment. [3]

Naidu, Kareppa, and Menon developed Dermato, a web-based application that employs deep learning for acne subtype classification (comedonal, pustular, rosacea, papules) and severity assessment. Their system processes user-uploaded images using a pipeline incorporating pre-processing, HSV color-based segmentation, and a fine-tuned InceptionV3 model. Dermato achieved an impressive classification accuracy of 99.88%, showcasing its potential to revolutionize acne diagnosis. While the reported accuracy is remarkable, further validation through clinical trials is essential to establish its reliability in real-world settings. Additionally, exploring the generalizability of the model to diverse skin tones and acne presentations would strengthen its clinical utility. [4]

Humayun, Malik, Belhaouri, and Kamel proposed a template matching approach for acne lesion localization using the chi-square test. To enhance accuracy, N-mean kernel filtering was applied as a preprocessing step. The method's performance was evaluated using precision, recall, accuracy, and F1 score. While the chi-square test effectively handles image variations, the authors acknowledge the challenges posed by cluttered backgrounds and noisy lesions, which can impact the method's overall performance. [5]

Ravi, K., Kiran, M. K., & Umadevi, V. provide a comprehensive overview of acne classification research. They highlight the predominant focus on detection and severity assessment, with limited exploration of detailed subtyping. The authors discuss the evolution of techniques from traditional image processing to deep learning-based approaches, emphasizing the challenges posed by imbalanced datasets and data scarcity. Data augmentation strategies, including both traditional methods and generative models like VAEs, have been explored to address these limitations. However, the optimal application of these techniques for acne classification remains an open research question. The study underscores the importance of careful architecture selection, hyperparameter optimization, and loss function design in achieving superior performance in this domain. [6]

Paluri Krishna Veni and Ashish Gupta introduced a novel hybrid deep learning model for acne severity classification. Their model effectively combines the robust feature extraction capabilities of VGG16 with the attention mechanism of a CBAM module, followed by a CapsNet for accurate classification. This innovative approach surpasses the limitations of traditional methods by achieving superior performance in distinguishing acne levels. [7]

Research Paper	Technique Used	Focus	Primary Insight
Automatic Segmentation and Classification of Eczema Skin Lesions Using Supervised Learning	Resnet – 18 (Residual Network with 18 layers)	Acne detection and classification: The system categorizes acne into four levels: normal, mild, moderate, and severe.	The model achieved a promising accuracy of 90% but its performance might be limited due to the specific dataset used. Future research should prioritize dataset diversity, acne localization refinement, and correlation analysis between acne severity scores and clinical assessments.
Dermato: A Deep Learning based Application for Acne Subtype and Severity Detection	HSV (Hue Saturation value) for image segmentation, Inception – V3 for Acne lesion classification.	Acne Subtype Classification : Categorizing acne into comedonal , pustular , rosacea and papules.	The Dermato system demonstrated exceptional classification accuracy (99.88%). While promising, the model requires validation through clinical trials to assess real-world reliability.
Localization of Acne Lesion through Template Matching	N-mean kernel for preprocessing. Using the chi-square test for similarity comparison	Acne lesion localization.	The chi-square test effectively handles image variations in acne lesion localization. Cluttered backgrounds and noisy lesions can negatively impact the method's performance.

Acne Classification using Deep Learning Models	VAE (Variational Autoencoder) deep learning architecture, ResNet -18	Acne Classification.	The performance of MobileNet, EfficientNet, and Inception ResNet models on the dataset of acne images was explored. Transfer learning was utilized to make the best use of the pre-trained models and various experiments were conducted to analyze their effectiveness.
Revolutionizing Acne Diagnosis with Hybrid Deep Learning Model	VGG16-CBAM-CapsNet Hybrid model.	.Acne Severity Classification.	The proposed hybrid model effectively combines different deep learning components for superior acne severity classification performance compared to traditional methods. The integration of VGG16 for feature extraction, CBAM for attention, and CapsNet for classification is a novel approach.

Table 1 : Overview of existing methods and technologies for Acne Lesion Detection and severity analysis

1.3 Research Gap

The extant literature showcases significant advancements in the application of machine learning (ML) and deep neural networks (DNNs) to diagnose, classify, and assess the severity of acne. However, despite these strides, there remain substantial gaps in research that hinder the development of robust and clinically applicable models, particularly in the

context of atopic dermatitis. The need for further exploration and refinement is evident to enhance the accuracy, generalizability, and clinical relevance of these methodologies.

Although the science of acne detection, prevention and treatment has been significantly advanced, a large space is still present in current literature. One of the main disadvantages is that there are not enough examples available to represent various groups and tend towards one group. This uniformity makes universal models difficult to fabricate which can detect acne accurately in all races and types of skin. In contrast to most prior studies conducted on data including mostly fair-skinned subjects, this research uses a well-rounded dataset with improved model generalizability and diagnostic accuracy.

Moreover, many deep learning models used for acne detection are black box in nature. Though effective at achieving state-of-the-art performance, the lack of interpretability in these models has led to concerns about reliability and limits their clinical translation. Explainable AI techniques must be used to provide an understanding of how the model produces its predictions in order improve trust and adoption by clinicians. Because of this feature, which will improve the user experience.

The absence of a full assessment for the degree of acne severity is another important research gap. Though there have been some studies to investigate this, a universally standardized desirable method has not yet developed. In addition to this, since acne is a dynamic skin condition, models that can follow disease evolution and potential treatment action needed over time are required. Addressing these limitations in future research may contribute to the development of acne management systems that are more evidence-based, reliable and clinically relevant.

In addition to these core areas, there is a need for research that investigates the integration of patient-reported outcomes (PROs) into acne assessment models. Incorporating patient-reported data can provide valuable insights into the impact of acne on quality of life and treatment satisfaction. Furthermore, exploring the potential of artificial intelligence in developing personalized treatment plans based on individual patient characteristics and acne progression is another promising avenue for future research.

By comprehensively addressing these research gaps, the field can advance towards the development of acne management systems that are not only highly accurate but also patient-centered, interpretable, and adaptable to diverse populations.

Feature / Gap	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**Proposed Solution**
Incorporate advanced deep learning techniques	Yes	Yes	Yes	Yes	Yes	Yes
Enhanced explainability	No	No	No	No	No	Yes
Diverse Datasets	No	Yes	Yes	Yes	Yes	Yes
Acne severity assessment	No	Yes	No	No	Yes	Yes
Healing Progress Tracking	No	No	No	No	No	yes

Table 2 : Comparison of the features addressed by the previous research studies and the features that will be covered in the proposed solution

1.4 Research Problem

Acne is a common chronic disorder affecting the hair follicle and sebaceous gland, in which there is expansion and blockage of the follicle and inflammation. Acne mostly appears among teenagers, but it can appear in any age range. The Acne lesions mostly appear on face but also appears on neck, shoulders and back. Accurate classification of acne severity plays a crucial role in selecting skincare products or treatments for patients. [8]

Acne severity has traditionally been judged by dermatologists using subjective evaluative methods. Manual inspection by a dermatologist is often time-consuming and may not always be efficiently used in high volume medical environments. This approach totally relies on the experience of a doctor when evaluating acne, which in the process of visual assessment often brings to inconsistency of a diagnosis and a treatment. Each stage of acne requires completely different approaches to treatment. Manual evaluative methods are also generally less efficient, especially when employed in high volume medical organizations. Finally, the condition being an integral element of a person's appearance, working with the problem of acne is particularly delicate in nature.

The restrictions of manual acne severity assessment demand objective, quantitative and reproducible ways. It is possible to improve diagnostic precision and treatment effectiveness by creating automated systems which use ML and AI technologies and could correctly classify acne severity. Moreover, this approach could provide patients with self-monitoring tools and could allow tracking acne progression overtime. Thus, it could be beneficial for patients' involvement in their skin health management.

1.5 Research Objectives

1.5.1 Main Objectives

The main purpose of this study is to elaborate on the innovative artificial intelligence and machine learning system that was manufactured to perceive, analyze, and track acne lesion progress accurately. The latter goal was achieved by creating a unique piece of technology that is capable of transforming acne management by offering dermatologists as well as acne patients an accurate and efficient treatment that is specifically tailored to an individual case of the disease.

Dermascope AI will provide a platform for doctors and patients to accurately diagnose acne lesions through advanced image processing models. These cutting-edge models will significantly enhance the accuracy of acne lesion identification compared to traditional methods. The need stems from the fact of the manual acne diagnosis low rate and the subjectivity of the procedure. This decision wastes precious time and is heavily reliant on the experienced specialist, which may lead to incorrect diagnoses because of different levels of dermatological knowledge of given healthcare professional. Notably, quick acne clearance is quite relevant for patients, so the wrongly issued treatment can exacerbate the condition. Conversely, when acne treatment is given in an appropriate stage of the condition, successfully treating it is relatively easy with correct treatment.

In addition, the system will identify the stage of severity of the Acne. This will help the doctors to provide proper severity stage related treatment. It is a must to give proper treatment for the concerned stage because the treatments are different from stage wise of the acne. However, if we can identify the correct stage, we can avoid unnecessary treatments or improper treatment which may result that the patient loses its time and money.

Beyond the other features, the system will also have a healing progress tracking facility. This aspect of the system will allow patients to track their acne issues and see improvement over time. patients can be impatient once they start treating acne, and they always want the treatment to clear up as soon as possible. However, how long it takes to heal acne might differ from one person to another because of the different healing frequencies of the emergence of acne. With the tracking system, patient will understand their clearing time compared to other people and accelerate return to their normal skin.

Ultimately, Dermascope AI seeks to revolutionize acne care by providing a reliable, accessible, and patient-centered technological solution.

1.5.2 Sub Objectives

- **Sub Objective 1 : Develop a Robust Acne Lesion Detection Model**

The first critical step in realizing "Dermascope AI" is producing an accurate high-functioning AI model that can identify and locate acne lesions within several dermatoscopic images. As a distinct component of AI, several techniques of image processing can be employed by the model to clear up as many acne lesions and reduce interference from noise. A variety of deep learning architectures, such as convolutional neural networks and object detection models, will be explored to optimize the model's ability to precisely locate acne lesions.

- **Sub Objective 2 : Implement Acne Severity Assessment**

In line with the lesion detection model, the system must classify acne lesions in terms of their severity. The implementation of the sub-objective will require the development of a classification model that will accurately identify the severity of such qualified criteria as size, shape, inflammation, and presence of lesions. To improve the precision of the classification, the model will rely on image features supported by pertinent patient information including their age, gender, and skin type. In addition, setting unambiguous and standardized criteria for severity is essential for consistent and accurate classification.

- **Sub Objective 3 : Enable Patient Acne Progression Tracking**

To trace the path of the disease, the system aims to include in its structure a feature that tracks the changes of the eruption of the wound. To do this, the model will compare the new images provided by the patient to the old, thus measuring how much the spots have decreased in size and how much the inflammation has diminished. The model will give the patients and the dermatologists a visual and numerical representation of the changes, which would make the evaluation of the effectiveness of the treatment easy.

- **Sub Objective 4 : Optimize User Interface and Clinical Integration**

By designing user-friendly interface, the system will be maximally used and its functionality will be successfully adopted in the clinical practice. Various training and supporting materials will be delivered to the healthcare providers. The design of the interface will be focused on the needs and characteristics of dermatologists and patients, and will originate to their overall experience from using the technology.

- **Sub Objective 5 : Create Comprehensive Patient Profiles**

To deliver personalized care, Dermascope AI will maintain detailed patient profiles. These profiles will store essential information including demographics, medical history, treatment plans, and a chronological record of image data. Robust data management and privacy measures will be implemented to protect patient information. By analyzing these profiles, the system can generate tailored treatment recommendations and valuable insights.

1.6 Requirement Gathering and Feasibility Study

1.6.1 Requirement Gathering and Analysis

During the requirement gathering and analysis phase the team thoroughly researched existing products in the dermatology field. This included acquiring a solid understanding regarding the existing studies, understanding new trends and conducting a comprehensive literature survey.

1.6.1.1 Functional Requirements

- **Core Functionalities**

- Image Acquisition: this system must accept high-resolution images of skin lesion capturing from various sources. It could be a direct upload from a user or integration with an existing repository of the images.

- Image Preprocessing: this system has to incorporate a variety of techniques concerning image processing. They include image enhancement to increase quality, normalization and prepare an image for further analysis to provide more accurate results.

- Acne Lesion Detection and Segmentation: The application must accurately identify and isolate acne lesions within the uploaded images, providing precise outlines of affected areas.

- Acne Severity Assessment: This application must provide both quantitative and qualitative assessment of acne severity and classify it according to clinical standards.

- Progress Tracking: The system has to compare the photos of the skin lesion provided by the customer and make a report on how it improves or guarantees.

- **User Interface and Experience**

  - User Registration and Authentication: Application Must Provide User Sign In or Login Process in Secure Manner

- Image Upload and Management: This allows users a simple way to upload, view and manage their image history.

- Severity Assessment Visualization: That the machine should offer clear and easy suitable diagnosis of acne breakouts severity just like, by way of visual terms or numerical scores.

- Progress Tracking Visualization: The system must be able to give visual aids on the progress of acne over identical locations throughout time, and monitor if treatments are effective.

- **Data Management and Storage**

  - Patient Data Storage: The application must securely store patient information, including medical history, treatment plans, and image data.
  - Data Privacy and Security: The system must adhere to strict data privacy regulations and implement robust security measures to protect user information.

1.6.1.2 Non-Functional Requirements

- **Performance**

  - Response Time: The system must provide quick response time to the user during image processing, analysis and report generation.

- Throughput: The system must handle to a certain extent requests that consists of multiple concurrent users and image processing tasks, without losing performance.

- Scalability: Scalable design shall allow increasing user and datasets whilst maintaining its performance.

- **Security and Privacy**

  - Data Confidentiality: Patient data, including images and medical information, must be protected with robust encryption and access controls.

  - Data Integrity: The system should maintain the accuracy and consistency of data throughout its lifecycle.

  - Data Availability: The system should ensure uninterrupted access to data and services.

- **Usability**

  - User Interface: The system should have a user-friendly interface that is intuitive and easy to navigate for both patients and healthcare providers.

  - Accessibility: The system should be accessible to users with disabilities, adhering to accessibility standards.

- **Reliability**

    - System Availability: The system should have high availability with minimal downtime to ensure uninterrupted service.

    - Error Handling: The system should gracefully handle errors and provide informative feedback to users.

- **Maintainability**

    - Code Quality: The system should be developed with clean, well-structured, and maintainable code.

1.6.1.3 Personal Requirements

- **Guidance from external supervisors on the dermatological domain**

This points to a need for dermatologist or skin specialist help. Their clinical expertise is invaluable for ensuring that the system remains relevant to real-life dermatology and as close a match as possible (in terms of acne morphology) with reality.

- **Guidance from supervisors on the data science domain**

Here we see a good example of why expert guidance is necessary in data science. DermaScope AI is a complex model trained using image analysis and machine learning therefore need good experienced data scientist as supervisor to assist in building resilient, reliable models.

1.6.2 Feasibility Study

1.6.2.1 Schedule Feasibility

A detailed project plan was created with the major milestones of the project and the expected deadlines to reach the goals defined. This project schedule timely completion and continuous monitoring for the progress of the component.

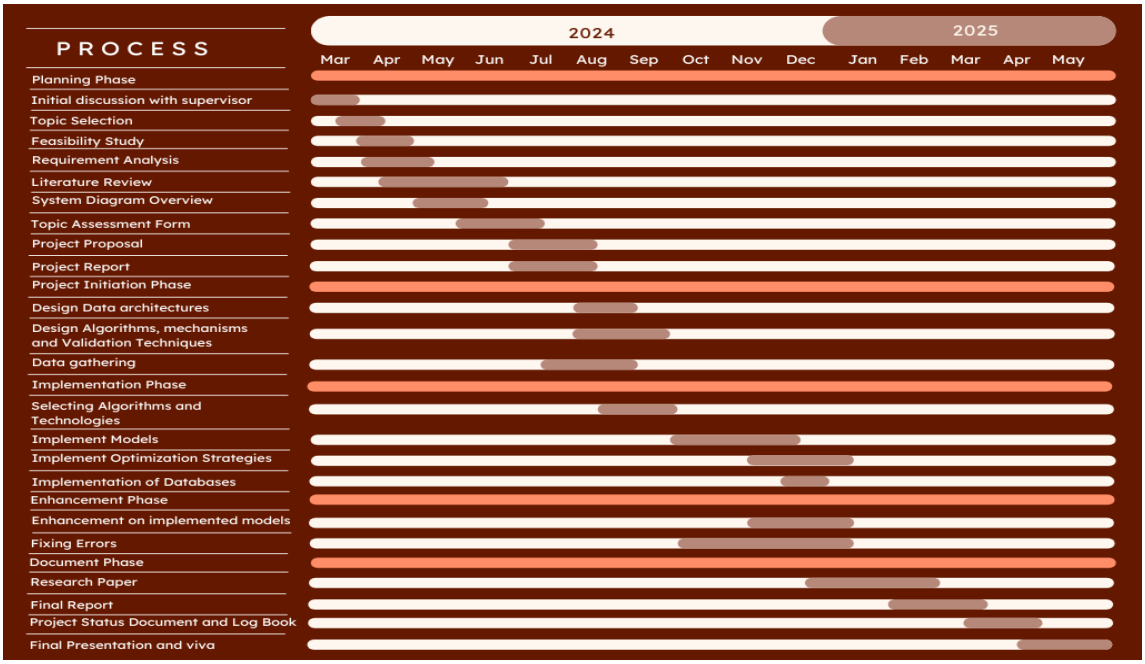


Figure 1-2 : Gantt Chart

Gantt Chart diagrammatically represents the plan timeline starting from March 2024 to May 2025. This holistic perspective covers the main stages of the project as well as what needs to be done within them one after another.

It starts with the first phase in March 2024 focusing on tasks like talking to supervisors and picking up a topic. Later steps include gathering and preparing data and the critical

step: implementing algorithms to build models with optimization strategies. The implementation schedule that spans the end of 2024 and beginning on the year 2025.

The project finishes with an iteration phase-performanc well as regarding some error corrections and documentation. The last few steps include preparing a research paper, writing a report and ending at the final presentation & viva by May 2025.

It allows for better project management as you can see all the task dependencies, durations and potential bottlenecks on a single visualization which is rather structured.

1.6.2.2 Technical Feasibility

Comprehensive research on the technical areas of expertise needed for the project, such as machine learning, convolutional neural networks, video and image processing, and other technologies was conducted. Further, the software and hardware requirements for the implementation process were taken into account.

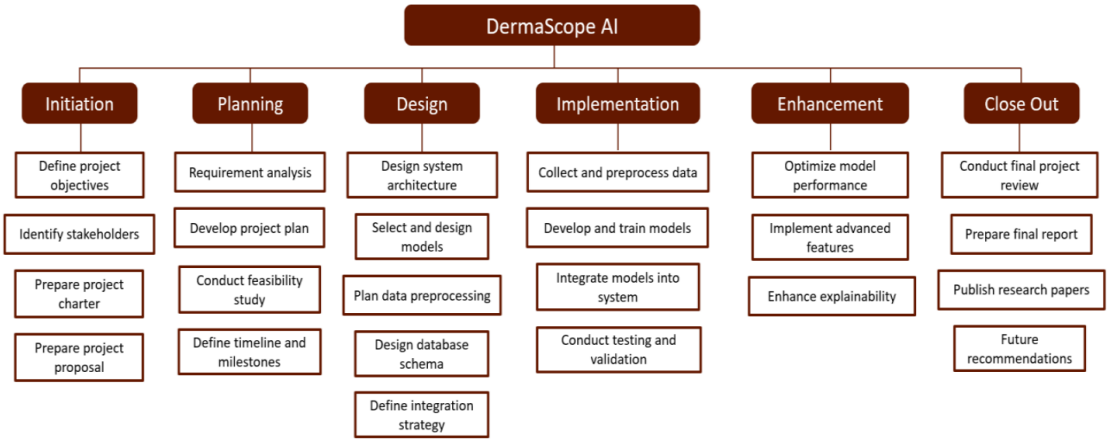


Figure 1-3 : Work Breakdown Structure

The Work Breakdown Structure outlines a structured approach to the DermaScope AI project, dividing it into six primary phases:

1. Initiation: This foundational phase involves defining project objectives, identifying stakeholders, and creating essential project documentation like the charter and proposal.

2. Planning: During this phase, the project team conducts a feasibility study, develops a detailed project plan, establishes timelines, and defines project milestones.

3. Design: Here, the system architecture is designed, models and databases are selected, and data preprocessing strategies are planned.

4. Implementation: This is the core development phase where data is collected and preprocessed, models are developed and trained, and system integration takes place. Rigorous testing and validation are also conducted in this stage.

5. Enhancement: Once the system is operational, the enhancement phase focuses on optimizing model performance, implementing advanced features, and improving the system's explainability.

6. Close-Out: This final phase involves conducting a comprehensive project review, preparing a final report, publishing research findings, and providing recommendations for future development.

1.6.2.3 Economic Feasibility

In the feasibility study economic feasibility was also considered. The cost of implementation and the benefits of the result of the study were assessed. Specifically, travelling expenses incurred for medical care with our doctors and to Kalubovila Hospital were evaluated.

	Amount
Travelling Cost	3000

Table 3 :Budget Allocation

1.6.2.4 Operational Feasibility

The operational feasibility of the research was done to assess the usefulness and to determine the practicality. This aimed to determine whether the users would find the final product useful and easy to use.

1.7 System Design

1.7.1 Overall System Design

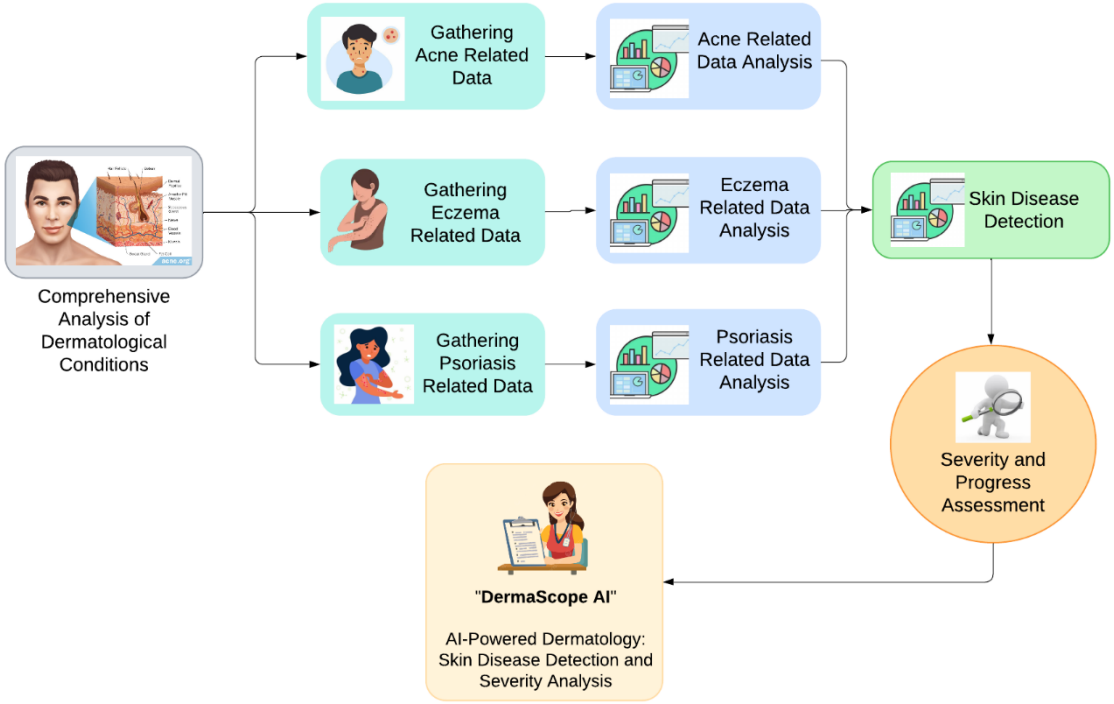


Figure 1.7.4 : Overall System Diagram

1.7.2 Acne Detection and Severity Assessment Component's Architecture Diagram

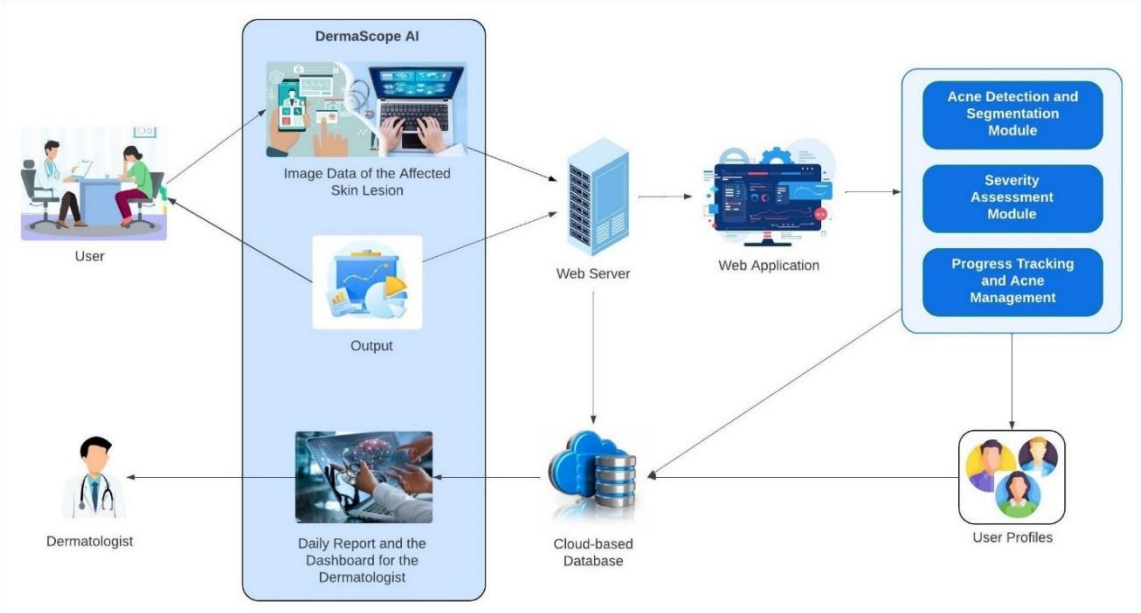


Figure 1.7.5 : Individual component system architecture diagram

DermaScope AI is a service designated to the sphere of acne's detection, segmentation and severity assessment. The service is using image analysis with artificial intelligence to provide the data for users. In a number of small steps let the user upload the acne lesion-localized within the same skin area. The uploaded images are analysed with the advanced hybrid image analysis model developed by the service. The method used by the service to assess the severity of acne was collecting facial acne lesion color which are used to analyze the stage of acne and spread are a of the lesion's area skin color gradation. Pre-processed results –which are the diagnosis of disease and severity of acne stages, by patient-, was stored in a server side of WEB application to make available communication of AI system part at client side with Web Application Interface. Patient data up to the level of uploaded images and the severity of the case along the way was kept by system.

Therefore, the Track of the rate of improvements made about the treatment on patients was kept and used to generate for patient personalized plans. The reports developed and generated by DermaScope AI for users. These Reports and dashboards shows the stage of Acne Severity which the user is in and its improvement, and during that time the extent of recommendation for treatment and their efficacy for patient.

1.8 Key Pillars of the Research Component

1.8.1 Acne Detection Model

Convolutional Neural Networks (CNNs) are at the core of the acne detection and analysis pipeline. These models are particularly well-suited for image-based tasks due to their ability to automatically learn hierarchical features from raw pixel data. The specific CNN architectures employed in this research include a custom-designed CNN architecture tailored to the specific characteristics of acne lesions and the requirements of the analysis task. Additionally, well-established deep CNN architectures like VGG16 and VGG19 are utilized, with pre-trained versions potentially fine-tuned for acne lesion detection to leverage their learned feature representations. EfficientNetB0, a more recent family of CNNs known for balancing accuracy and computational efficiency, is also employed. Siamese networks are used for feature extraction, learning discriminative embeddings for different types and severities of acne lesions. Finally, Residual Networks (ResNets), another popular deep CNN architecture known for its performance in image recognition, is used for acne lesion analysis.

1.8.2 Lesion Localization with OpenCV

OpenCV (Open-Source Computer Vision Library) [9] is a powerful library for image processing and computer vision tasks. For this research, OpenCV plays a crucial role in

preprocessing images through techniques like noise reduction, image enhancement, and resizing. It is also used for Region of Interest (ROI) extraction, identifying and extracting regions containing potential acne lesions using methods like thresholding, contour detection, or blob analysis. Furthermore, OpenCV facilitates geometric analysis, enabling the measurement of lesion size, shape, and other geometric properties, which can be valuable features for classification and severity assessment.

1.8.3 Acne Severity Assessment

Beyond simple detection, this research aims to assess the severity of acne lesions, classifying them into different severity grades based on visual characteristics. The models utilized for this task include EfficientNetB0, CNN, and VGG16, potentially fine-tuned or adapted to classify acne lesions into different severity levels. ResNet50V2, a variant of the ResNet architecture known for its improved performance and efficiency, is also used to learn the subtle visual differences that distinguish between different acne severity grades.

2 Methodology

2.1 Methodology – Acne Detection

2.1.1 Data Collection

initial collection of images collected from the publicly accessible DermNet website and the Kaggle platform, a critical validation step was undertaken to ensure the suitability and reliability of the data for this research focused on automated acne lesion analysis. Recognizing the importance of clinical accuracy in medical image analysis, two external

supervisors, both established dermatologists with expertise in diagnosing and treating skin conditions, were engaged to meticulously review the collected image dataset. This thorough review process involved assessing the quality of the images, confirming the accurate identification and labeling of acne lesions, and ensuring that the images represented a diverse range of acne types and severities. The dermatologists' validation served as a crucial safeguard, confirming that the dataset met the necessary clinical standards and was appropriate for training and evaluating the developed artificial intelligence models. This rigorous validation procedure strengthens the credibility of the research by establishing a high-quality, clinically relevant dataset as the foundation for subsequent analysis and the development of automated diagnostic tools for acne.

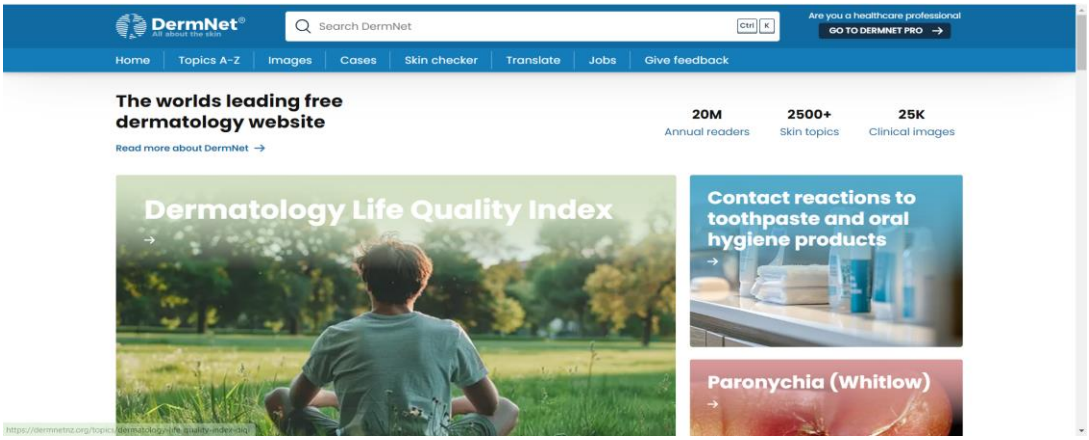


Image 1 : DermNet Website for Data Collection

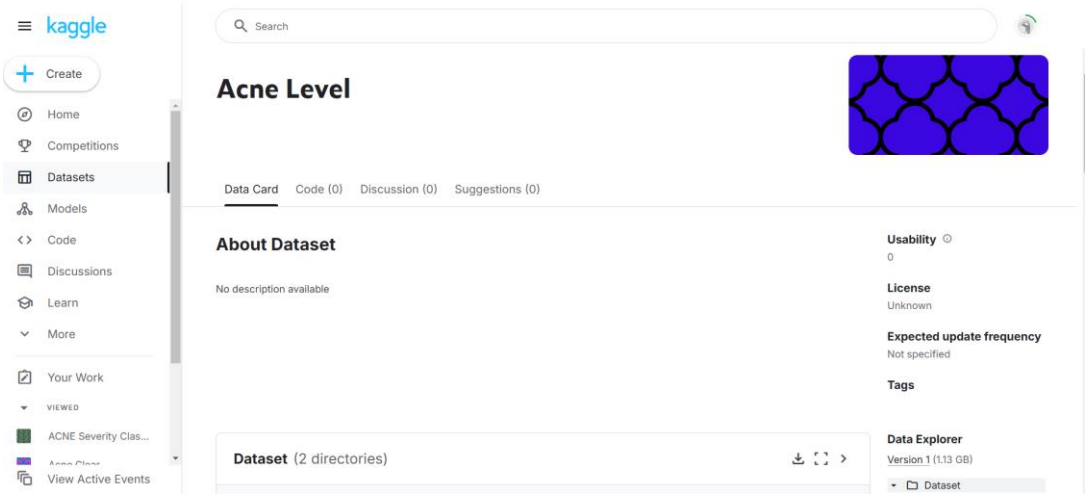
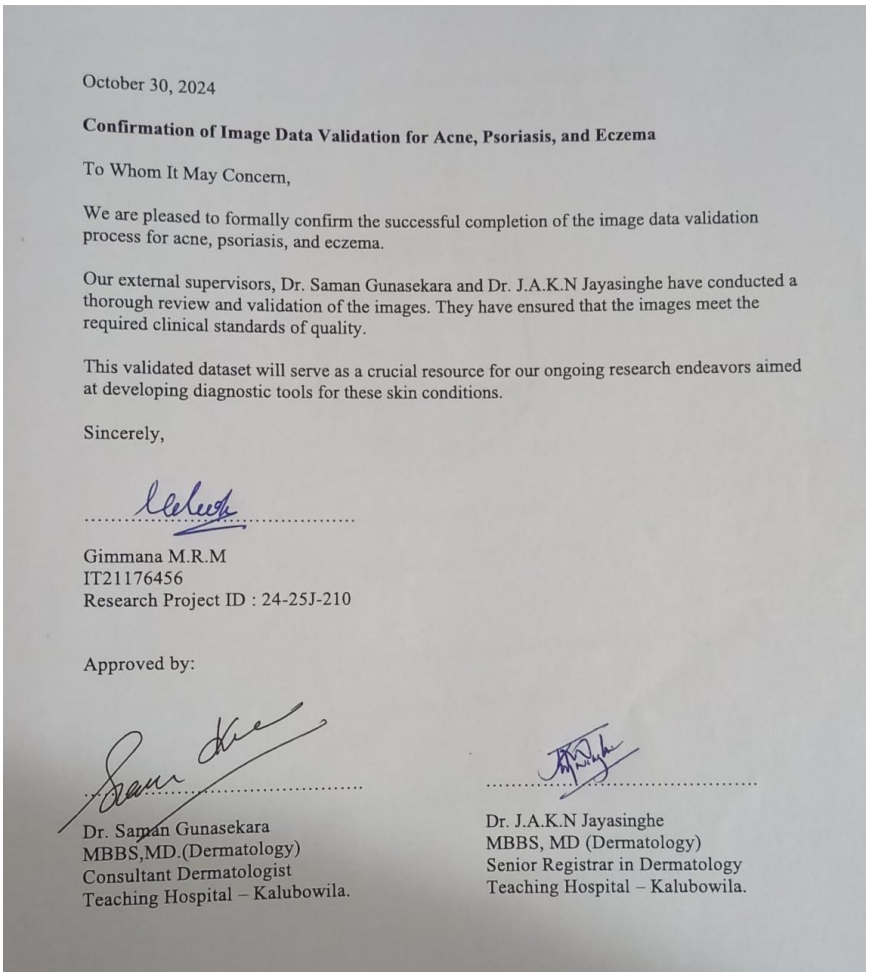


Image 2 : DermNet Website for Data Collection



October 30, 2024

Confirmation of Image Data Validation for Acne, Psoriasis, and Eczema

To Whom It May Concern,

We are pleased to formally confirm the successful completion of the image data validation process for acne, psoriasis, and eczema.

Our external supervisors, Dr. Saman Gunasekara and Dr. J.A.K.N Jayasinghe have conducted a thorough review and validation of the images. They have ensured that the images meet the required clinical standards of quality.

This validated dataset will serve as a crucial resource for our ongoing research endeavors aimed at developing diagnostic tools for these skin conditions.

Sincerely,

.....

Gimmana M.R.M
IT21176456
Research Project ID : 24-25J-210

Approved by:

.....

Dr. Saman Gunasekara
MBBS,MD.(Dermatology)
Consultant Dermatologist
Teaching Hospital – Kalubowila.

.....

Dr. J.A.K.N Jayasinghe
MBBS, MD (Dermatology)
Senior Registrar in Dermatology
Teaching Hospital – Kalubowila.

Image 3 : Proof of Data Valdation with External Supervisors

2.1.2 Data Preprocessing

The development of an effective acne detection model necessitated a carefully designed data preprocessing and augmentation pipeline to ensure robust performance across varying clinical presentations. The preprocessing methodology was structured to address key challenges in dermatological image analysis while optimizing the dataset for deep learning applications.

At the core of the preprocessing workflow was the standardization of input images to ensure consistency in model training. All acne images were systematically resized to dimensions of 224×224 pixels, a resolution chosen to balance computational efficiency with sufficient detail preservation for lesion identification. This resizing operation was accompanied by pixel value normalization, where raw image intensities were scaled to a [0,1] range through division by 255. This normalization step proved crucial for stabilizing gradient updates during the subsequent neural network training process. The conversion from BGR to RGB color space addressed potential inconsistencies arising from different image acquisition systems, ensuring color representation uniformity across the dataset.

The augmentation strategy incorporated both geometric transformations and color-space manipulations to comprehensively address the variability inherent in clinical dermatological images. The geometric augmentation pipeline, implemented through Keras' ImageDataGenerator, introduced controlled variations in image perspective and orientation. A 20-degree rotation range accommodated natural variations in camera angle during image capture, while width and height shifts of 20% accounted for minor framing inconsistencies. Shear transformations at 20% intensity and zoom variations within a 20% range further enhanced the model's ability to recognize lesions under different presentation conditions. Horizontal flipping was employed to enforce left-right symmetry awareness, particularly valuable given the bilateral nature of many acne presentations. The 'nearest' fill mode selection ensured that these transformations did not introduce artificial artifacts at image boundaries.

Complementing the geometric augmentations, a sophisticated color-based processing approach was developed to specifically target the characteristic features of acne lesions. The implementation of HSV color space conversion facilitated more accurate detection of erythematous regions associated with inflammatory acne. Through the application of dual-range thresholding (0-10° and 160-180° hue ranges), the system could reliably identify both bright red active lesions and deeper inflammatory regions. Subsequent morphological processing, including opening and closing operations with a 5×5 kernel,

served to refine the lesion masks by eliminating noise and filling discontinuities in detected regions. This color-based processing not only supported data augmentation but also enabled precise bounding box generation for lesion localization tasks.

The preprocessing pipeline incorporated intelligent dataset partitioning through stratified sampling, maintaining the original distribution of acne severity levels across both training and validation subsets. This approach guarded against potential biases that might arise from uneven class representation. The implementation included comprehensive logging mechanisms to track all preprocessing operations, facilitating reproducibility and error diagnosis. These logs captured critical metadata including processing timestamps, image sources, transformation parameters, and any encountered exceptions.

The augmentation and preprocessing methodology demonstrated particular effectiveness in addressing several key challenges in acne image analysis. Variations in lighting conditions, a common issue in clinical photography, were mitigated through the combined effects of color normalization and HSV-based processing. Differences in patient skin tones were accommodated by the color-agnostic nature of the geometric augmentations coupled with the adaptive thresholding in HSV space. The preprocessing pipeline's ability to generate diverse training examples from limited original data proved invaluable in preventing overfitting, a significant concern given the relatively small size of many medical image datasets.

This comprehensive preprocessing framework supported the subsequent development of a deep learning system capable of both acne severity classification and precise lesion localization. The attention to both geometric and color-based transformations resulted in a model that maintained diagnostic accuracy across a wide range of image acquisition conditions and patient presentations. The success of this approach underscores the importance of tailored preprocessing in medical computer vision applications, where biological variability and imaging inconsistencies present unique challenges not typically encountered in general image recognition tasks. Future refinements could explore adaptive augmentation strategies that dynamically adjust parameters based on individual

image characteristics, potentially offering further improvements in model performance and generalization capability.

Image 4 : Color based Augmentation

Image 5 : Data Augmentation using ImageDataGenerator

2.1.3 Model Building and Training

Six deep learning architectures, VGG16, CNN, VGG19, EfficientNetB0, Siamese network, and ResNet were rigorously evaluated for the acne detection task, with the optimal model being selected based on comparative performance analysis.

Approach 1: CNN

```
model_cnn = Sequential()

model_cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding='same',
                                     activation='relu',input_shape=[224,224,3],kernel_regularizer=tf.keras.regularizers.l1(0.001)))
model_cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu'))
model_cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

model_cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding='same',
                                     activation='relu',kernel_regularizer=tf.keras.regularizers.l1(0.001)))
model_cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,activation='relu'))
model_cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

model_cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,padding='same',
                                     activation='relu',kernel_regularizer=tf.keras.regularizers.l1(0.001)))
model_cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,activation='relu'))
model_cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

model_cnn.add(tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='same',
                                     activation='relu',kernel_regularizer=tf.keras.regularizers.l1(0.001)))
model_cnn.add(tf.keras.layers.Conv2D(filters=256,kernel_size=3,activation='relu'))
model_cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))

model_cnn.add(tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',
                                     activation='relu',kernel_regularizer=tf.keras.regularizers.l1(0.001)))
model_cnn.add(tf.keras.layers.Conv2D(filters=512,kernel_size=3,activation='relu'))
model_cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
model_cnn.add(tf.keras.layers.Dropout(0.25))

model_cnn.add(tf.keras.layers.Flatten())

model_cnn.add(tf.keras.layers.Dense(units=1500,activation='relu'))

model_cnn.add(tf.keras.layers.Dropout(0.4)) #To avoid overfitting

#Output Layer
model_cnn.add(tf.keras.layers.Dense(units=1,activation='sigmoid'))
```

```python
# Import EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping

# Set up EarlyStopping
early_stopping = EarlyStopping(
    monitor='val_loss',  # Metric to monitor
    patience=10,         # Number of epochs with no improvement after which training will be stopped
    restore_best_weights=True  # Restore the best weights from the epoch with the lowest validation loss
)
```

```python
# Compile the model (pass callbacks as a separate argument)
model_cnn.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

```python
model_cnn.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 32)	896
conv2d_1 (Conv2D)	(None, 222, 222, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_2 (Conv2D)	(None, 111, 111, 64)	18,496
conv2d_3 (Conv2D)	(None, 109, 109, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_4 (Conv2D)	(None, 54, 54, 128)	73,856
conv2d_5 (Conv2D)	(None, 52, 52, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
conv2d_6 (Conv2D)	(None, 26, 26, 256)	295,168
conv2d_7 (Conv2D)	(None, 24, 24, 256)	590,080
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 256)	0

max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 256)	0
conv2d_8 (Conv2D)	(None, 12, 12, 512)	1,180,160
conv2d_9 (Conv2D)	(None, 10, 10, 512)	2,359,808
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 512)	0
dropout (Dropout)	(None, 5, 5, 512)	0
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 1500)	19,201,500
dropout_1 (Dropout)	(None, 1500)	0
dense_1 (Dense)	(None, 1)	1,501

Total params: 23,915,225 (91.23 MB)

Trainable params: 23,915,225 (91.23 MB)

Non-trainable params: 0 (0.00 B)

```
epochs_cnn = 70
# Train the model with the EarlyStopping callback
history_cnn = model_cnn.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=epochs_cnn,
    callbacks=[early_stopping]  # Add the early stopping callback here
)
```

```
Epoch 3/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 265s 5s/step - accuracy: 0.8805 - loss: 13.8195 - val_accuracy: 0.9060 - val_loss: 10.7192
Epoch 4/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 268s 5s/step - accuracy: 0.8965 - loss: 9.8720 - val_accuracy: 0.9181 - val_loss: 7.4775
Epoch 5/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 268s 5s/step - accuracy: 0.9090 - loss: 6.8621 - val_accuracy: 0.9108 - val_loss: 5.2473
Epoch 6/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 559s 11s/step - accuracy: 0.9212 - loss: 4.8591 - val_accuracy: 0.9253 - val_loss: 3.9330
Epoch 7/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 194s 4s/step - accuracy: 0.9359 - loss: 3.6741 - val_accuracy: 0.9301 - val_loss: 3.1426
Epoch 8/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 179s 3s/step - accuracy: 0.9168 - loss: 2.9733 - val_accuracy: 0.9060 - val_loss: 2.6186
Epoch 9/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 176s 3s/step - accuracy: 0.9374 - loss: 2.4610 - val_accuracy: 0.9229 - val_loss: 2.1823
Epoch 10/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 175s 3s/step - accuracy: 0.9367 - loss: 2.0866 - val_accuracy: 0.9205 - val_loss: 1.9065
Epoch 11/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 174s 3s/step - accuracy: 0.9440 - loss: 1.8057 - val_accuracy: 0.9325 - val_loss: 1.6872
Epoch 12/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 179s 3s/step - accuracy: 0.9402 - loss: 1.6034 - val_accuracy: 0.9181 - val_loss: 1.5237
Epoch 13/70
...
Epoch 69/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 167s 3s/step - accuracy: 0.9703 - loss: 0.2829 - val_accuracy: 0.9398 - val_loss: 0.4412
Epoch 70/70
52/52 ━━━━━━━━━━━━━━━━━━━━ 169s 3s/step - accuracy: 0.9834 - loss: 0.2538 - val_accuracy: 0.9325 - val_loss: 0.4502
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Approach 2: VGG16

```
#Fine-tuning
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 101s 2us/step
```

```
fine_tune_at = 10
```

```
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

```python
# 3. Architecture Optimization
model_vgg16 = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),  # Increased units
    BatchNormalization(),           # Added Batch Normalization
    Dropout(0.3),                   # Adjusted dropout rate
    Dense(128, activation='relu'),  # Added another Dense layer
    BatchNormalization(),           # Added Batch Normalization
    Dropout(0.2),                   # Added dropout
    Dense(1, activation='sigmoid')
])
```

```python
model_vgg16.compile(optimizer=Adam(learning_rate=1e-4),  # Higher learning rate for new layers
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
```

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
flatten_1 (Flatten)	(None, 25088)	0
dense_2 (Dense)	(None, 256)	6,422,784
batch_normalization (BatchNormalization)	(None, 256)	1,024
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32,896
batch_normalization_1 (BatchNormalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 1)	129

```python
#Hyperparameter Tuning (using ReduceLROnPlateau)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=1e-6)
```

```python
epochs_vgg16 = 50 # Increased epochs
history_vgg16 = model_vgg16.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=epochs_vgg16,
    callbacks=[reduce_lr]  # Added learning rate scheduler
)
```

```
52/52 ━━━━━━━━━━━━━━━━━━━━ 869s 17s/step - accuracy: 0.9727 - loss: 0.0814 - val_accuracy: 0.7277 - val_loss: 1.0981 - learning_rate: 1.0000e-04
Epoch 3/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 923s 18s/step - accuracy: 0.9789 - loss: 0.0572 - val_accuracy: 0.5470 - val_loss: 1.3435 - learning_rate: 1.0000e-04
Epoch 4/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 797s 15s/step - accuracy: 0.9856 - loss: 0.0463 - val_accuracy: 0.9807 - val_loss: 0.0602 - learning_rate: 1.0000e-04
Epoch 5/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 811s 16s/step - accuracy: 0.9906 - loss: 0.0298 - val_accuracy: 0.9928 - val_loss: 0.0327 - learning_rate: 1.0000e-04
Epoch 6/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 860s 17s/step - accuracy: 0.9950 - loss: 0.0194 - val_accuracy: 0.9880 - val_loss: 0.0393 - learning_rate: 1.0000e-04
Epoch 7/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 811s 16s/step - accuracy: 0.9961 - loss: 0.0216 - val_accuracy: 0.9855 - val_loss: 0.0456 - learning_rate: 1.0000e-04
Epoch 8/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 760s 15s/step - accuracy: 0.9964 - loss: 0.0166 - val_accuracy: 0.9928 - val_loss: 0.0488 - learning_rate: 1.0000e-04
Epoch 9/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 757s 15s/step - accuracy: 0.9980 - loss: 0.0089 - val_accuracy: 0.9880 - val_loss: 0.0421 - learning_rate: 2.0000e-05
Epoch 10/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 751s 14s/step - accuracy: 0.9980 - loss: 0.0097 - val_accuracy: 0.9904 - val_loss: 0.0377 - learning_rate: 2.0000e-05
Epoch 11/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 782s 15s/step - accuracy: 0.9998 - loss: 0.0038 - val_accuracy: 0.9904 - val_loss: 0.0280 - learning_rate: 2.0000e-05
Epoch 12/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 792s 15s/step - accuracy: 0.9990 - loss: 0.0057 - val_accuracy: 0.9928 - val_loss: 0.0429 - learning_rate: 2.0000e-05
Epoch 13/100
...
Epoch 99/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 614s 12s/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.9904 - val_loss: 0.0361 - learning_rate: 1.0000e-06
Epoch 100/100
52/52 ━━━━━━━━━━━━━━━━━━━━ 732s 14s/step - accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.9928 - val_loss: 0.0376 - learning_rate: 1.0000e-06
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Approach 3: VGG19

```
base_model_vgg19 = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model_vgg19.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 160s 2us/step
```

```
model_vgg19 = Sequential([
    base_model_vgg19,
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
```

```
model_vgg19.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
model_vgg19.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 7, 7, 512)	20,024,384
flatten_2 (Flatten)	(None, 25088)	0
dense_5 (Dense)	(None, 128)	3,211,392
dropout_4 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 1)	129

Total params: 23,235,905 (88.64 MB)

Trainable params: 3,211,521 (12.25 MB)

Non-trainable params: 20,024,384 (76.39 MB)

```python
epochs_vgg19 = 50
history_vgg19 = model_vgg19.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=epochs_vgg19
)
```

```
Epoch 3/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 507s 10s/step - accuracy: 0.9889 - loss: 0.0386 - val_accuracy: 0.9831 - val_loss: 0.0622
Epoch 4/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 477s 9s/step - accuracy: 0.9959 - loss: 0.0200 - val_accuracy: 0.9711 - val_loss: 0.0833
Epoch 5/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 461s 9s/step - accuracy: 0.9972 - loss: 0.0189 - val_accuracy: 0.9783 - val_loss: 0.0612
Epoch 6/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 531s 10s/step - accuracy: 0.9925 - loss: 0.0209 - val_accuracy: 0.9807 - val_loss: 0.0459
Epoch 7/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 449s 9s/step - accuracy: 0.9982 - loss: 0.0070 - val_accuracy: 0.9783 - val_loss: 0.0497
Epoch 8/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 494s 10s/step - accuracy: 0.9979 - loss: 0.0067 - val_accuracy: 0.9783 - val_loss: 0.0557
Epoch 9/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 663s 13s/step - accuracy: 0.9981 - loss: 0.0051 - val_accuracy: 0.9855 - val_loss: 0.0495
Epoch 10/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 792s 15s/step - accuracy: 0.9998 - loss: 0.0021 - val_accuracy: 0.9831 - val_loss: 0.0674
Epoch 11/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 774s 15s/step - accuracy: 1.0000 - loss: 0.0041 - val_accuracy: 0.9807 - val_loss: 0.0546
Epoch 12/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 768s 15s/step - accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.9807 - val_loss: 0.0594
Epoch 13/50
...
Epoch 49/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 743s 14s/step - accuracy: 1.0000 - loss: 6.5053e-05 - val_accuracy: 0.9807 - val_loss: 0.0681
Epoch 50/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 729s 14s/step - accuracy: 1.0000 - loss: 1.8367e-04 - val_accuracy: 0.9807 - val_loss: 0.0718
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Approach 4 : ResNet

```
base_model_resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model_resnet.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50 we
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 209s 2us/step
```

+ Code + Markdown

```
model_resnet = Sequential([
    base_model_resnet,
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
flatten_3 (Flatten)	(None, 100352)	0
dense_7 (Dense)	(None, 128)	12,845,184
dropout_5 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 1)	129

Total params: 36,433,025 (138.98 MB)

Trainable params: 12,845,313 (49.00 MB)

Non-trainable params: 23,587,712 (89.98 MB)

```python
epochs_resnet = 50
history_resnet = model_resnet.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=epochs_resnet
)
```

```
Epoch 1/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 178s 3s/step - accuracy: 0.5063 - loss: 3.0477 - val_accuracy: 0.5157 - val_loss: 0.6906
Epoch 2/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 164s 3s/step - accuracy: 0.5125 - loss: 0.6918 - val_accuracy: 0.6024 - val_loss: 0.6610
Epoch 3/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 171s 3s/step - accuracy: 0.6084 - loss: 0.6555 - val_accuracy: 0.7831 - val_loss: 0.6039
```

```
Epoch 3/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 171s 3s/step - accuracy: 0.6084 - loss: 0.6555 - val_accuracy: 0.7831 - val_loss: 0.6039
Epoch 4/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 173s 3s/step - accuracy: 0.6507 - loss: 0.5987 - val_accuracy: 0.8627 - val_loss: 0.5219
Epoch 5/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 169s 3s/step - accuracy: 0.7301 - loss: 0.5508 - val_accuracy: 0.7855 - val_loss: 0.5192
Epoch 6/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.7727 - loss: 0.4875 - val_accuracy: 0.8843 - val_loss: 0.3978
Epoch 7/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 138s 3s/step - accuracy: 0.7964 - loss: 0.4280 - val_accuracy: 0.7711 - val_loss: 0.4817
Epoch 8/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 152s 3s/step - accuracy: 0.7148 - loss: 0.4862 - val_accuracy: 0.8795 - val_loss: 0.3179
Epoch 9/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 143s 3s/step - accuracy: 0.8009 - loss: 0.3852 - val_accuracy: 0.8795 - val_loss: 0.3068
Epoch 10/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 145s 3s/step - accuracy: 0.8076 - loss: 0.3578 - val_accuracy: 0.8843 - val_loss: 0.3032
Epoch 11/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 128s 2s/step - accuracy: 0.7972 - loss: 0.3657 - val_accuracy: 0.8771 - val_loss: 0.3272
Epoch 12/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 125s 2s/step - accuracy: 0.8242 - loss: 0.3438 - val_accuracy: 0.8988 - val_loss: 0.2494
Epoch 13/50
...
Epoch 49/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 130s 3s/step - accuracy: 0.8720 - loss: 0.2483 - val_accuracy: 0.9349 - val_loss: 0.1932
Epoch 50/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 133s 3s/step - accuracy: 0.8302 - loss: 0.2898 - val_accuracy: 0.9229 - val_loss: 0.1940
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Approach 5 : EfficientNetB0

```
base_model_efficientnet = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model_efficientnet.trainable = False
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 ━━━━━━━━━━━━━━━━━━━━ 14s 1us/step
```

```
model_efficientnet = Sequential([
    base_model_efficientnet,
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
```

```
model_efficientnet.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
model_efficientnet.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571
flatten_5 (Flatten)	(None, 62720)	0
dense_12 (Dense)	(None, 128)	8,028,288
dropout_8 (Dropout)	(None, 128)	0
dense_13 (Dense)	(None, 1)	129

Total params: 12,077,988 (46.07 MB)

Trainable params: 8,028,417 (30.63 MB)

Non-trainable params: 4,049,571 (15.45 MB)

```python
epochs_efficientnet = 40
history_efficientnet = model_efficientnet.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=epochs_efficientnet
)
```

```
52/52 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.4944 - loss: 0.7075 - val_accuracy: 0.5108 - val_loss: 0.6931
Epoch 3/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 64s 1s/step - accuracy: 0.4941 - loss: 0.6932 - val_accuracy: 0.5108 - val_loss: 0.6931
Epoch 4/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 65s 1s/step - accuracy: 0.4937 - loss: 0.6932 - val_accuracy: 0.5108 - val_loss: 0.6931
Epoch 5/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.5011 - loss: 0.6932 - val_accuracy: 0.5108 - val_loss: 0.6931
Epoch 6/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 77s 1s/step - accuracy: 0.5008 - loss: 0.6931 - val_accuracy: 0.5108 - val_loss: 0.6931
Epoch 7/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 68s 1s/step - accuracy: 0.5171 - loss: 0.6932 - val_accuracy: 0.5108 - val_loss: 0.6931
Epoch 8/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 72s 1s/step - accuracy: 0.4747 - loss: 0.6932 - val_accuracy: 0.4892 - val_loss: 0.6932
Epoch 9/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 68s 1s/step - accuracy: 0.4784 - loss: 0.6932 - val_accuracy: 0.4892 - val_loss: 0.6932
Epoch 10/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.4985 - loss: 0.6932 - val_accuracy: 0.4892 - val_loss: 0.6932
Epoch 11/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 56s 1s/step - accuracy: 0.4806 - loss: 0.7115 - val_accuracy: 0.4892 - val_loss: 0.6932
Epoch 12/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 63s 1s/step - accuracy: 0.4858 - loss: 0.6932 - val_accuracy: 0.4892 - val_loss: 0.6932
Epoch 13/40
...
Epoch 39/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - accuracy: 0.4818 - loss: 0.6934 - val_accuracy: 0.4892 - val_loss: 0.6933
Epoch 40/40
52/52 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - accuracy: 0.4907 - loss: 0.6933 - val_accuracy: 0.4892 - val_loss: 0.6933
```

Approach 6 : Siamese Neral Networks

```python
import os
import random
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import layers, Model, Input
from tensorflow.keras.callbacks import Callback
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

```python
# Define constants
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 16
EPOCHS = 20

# Load the dataset
path = "../SiameseD"
data = {"imgpath": [], "labels": []}
categories = os.listdir(path)
for folder in categories:
    folderpath = os.path.join(path, folder)
    filelist = os.listdir(folderpath)
    for file in filelist:
        fpath = os.path.join(folderpath, file)
        data["imgpath"].append(fpath)
        data["labels"].append(folder)
df = pd.DataFrame(data)
```

```python
# Split the dataset into train, validation, and test sets
train_df, temp_df = train_test_split(df, train_size=0.70, shuffle=True, random_state=124)
valid_df, test_df = train_test_split(temp_df, train_size=0.70, shuffle=True, random_state=124)
```

```python
# Encode labels
label_encoder = LabelEncoder()
train_labels = label_encoder.fit_transform(train_df["labels"])
valid_labels = label_encoder.transform(valid_df["labels"])
test_labels = label_encoder.transform(test_df["labels"])
```

```python
# Convert labels to one-hot encoded vectors
num_classes = len(set(train_labels))
train_labels = to_categorical(train_labels, num_classes=num_classes)
valid_labels = to_categorical(valid_labels, num_classes=num_classes)
test_labels = to_categorical(test_labels, num_classes=num_classes)
```

+ Code + Markdown

```python
# Load and preprocess images
train_images = load_images(train_df["imgpath"])
valid_images = load_images(valid_df["imgpath"])
test_images = load_images(test_df["imgpath"])
```

```python
# Convert labels back to integers for pairing
train_labels_int = np.argmax(train_labels, axis=1)
valid_labels_int = np.argmax(valid_labels, axis=1)
test_labels_int = np.argmax(test_labels, axis=1)
```

```python
# Function to create pairs of images and labels
#Positive pairs: Two images from the same class (label = 1).
#Negative pairs: Two images from different classes (label = 0)
def create_pairs(images, labels, num_pairs=1000):
    pairs = []
    pair_labels = []

    num_classes = len(set(labels))
    class_indices = [np.where(labels == i)[0] for i in range(num_classes)]

    for _ in range(num_pairs):
        # Randomly select two images from the same class
        class_idx = random.randint(0, num_classes - 1)
        idx1, idx2 = random.sample(list(class_indices[class_idx]), 2)
        pairs.append([images[idx1], images[idx2]])
        pair_labels.append(1)  # Same class

        # Randomly select two images from different classes
        class_idx1, class_idx2 = random.sample(range(num_classes), 2)
        idx1 = random.choice(class_indices[class_idx1].tolist())
        idx2 = random.choice(class_indices[class_idx2].tolist())
        pairs.append([images[idx1], images[idx2]])
        pair_labels.append(0)  # Different classes

    return np.array(pairs), np.array(pair_labels)
```

```python
# Create pairs for training, validation, and testing
train_pairs, train_pair_labels = create_pairs(train_images, train_labels_int, num_pairs=1250)
valid_pairs, valid_pair_labels = create_pairs(valid_images, valid_labels_int, num_pairs=500)
test_pairs, test_pair_labels = create_pairs(test_images, test_labels_int, num_pairs=500)
```

```python
# Define the base network (shared between the two branches)
def create_base_network(input_shape):
    input = Input(shape=input_shape)

    x = layers.Conv2D(64, (3, 3), activation='relu')(input)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(256, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Flatten()(x)
    x = layers.Dense(256, activation='relu')(x)

    return Model(input, x)
```

```python
# Define the Siamese network
input_shape = (IMAGE_SIZE[0], IMAGE_SIZE[1], 3)
base_network = create_base_network(input_shape)

input_a = Input(shape=input_shape)
input_b = Input(shape=input_shape)

output_a = base_network(input_a)
output_b = base_network(input_b)

distance = layers.Lambda(lambda x: tf.abs(x[0] - x[1]))([output_a, output_b])
output = layers.Dense(1, activation='sigmoid')(distance)

siamese_model = Model([input_a, input_b], output)

# Compile the Siamese network
# Adam optimizer and binary cross-entropy loss, which is suitable for binary classification tasks.
siamese_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
WARNING:tensorflow:From e:\sliit\Year 4\Research Project\DEMO\.venv\Lib\site-packages\keras\src\backend\tensc
```

```python
# Custom callback to stop training if validation accuracy reaches 90%
# The MyCallback class can be used to stop training early if a certain validation accuracy is achieved.
class MyCallback(Callback):
    def on_epoch_end(self, epoch, logs=None):
        val_accuracy = logs.get("val_accuracy")
        print(f"Validation Accuracy at epoch {epoch + 1}: {val_accuracy}")
        # if val_accuracy >= 0.9:
        #     print("\nReached 90% accuracy! Stopping training...")
        #     self.model.stop_training = True
```

```python
# Train the Siamese network
history = siamese_model.fit(
    [train_pairs[:, 0], train_pairs[:, 1]], train_pair_labels,
    validation_data=([valid_pairs[:, 0], valid_pairs[:, 1]], valid_pair_labels),
    batch_size=BATCH_SIZE, epochs=EPOCHS,
    callbacks=[MyCallback()]
)
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 485s 3s/step - accuracy: 0.7637 - loss: 0.5327 - val_accuracy: 0.8910 - val_loss: 0.2915
Epoch 2/20
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9241 - loss: 0.2566Validation Accuracy at epoch 2: 0.9120000004768372
157/157 ━━━━━━━━━━━━━━━━━━━━ 372s 2s/step - accuracy: 0.9240 - loss: 0.2568 - val_accuracy: 0.9120 - val_loss: 0.2669
Epoch 3/20
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9619 - loss: 0.1579Validation Accuracy at epoch 3: 0.9279999732971191
157/157 ━━━━━━━━━━━━━━━━━━━━ 390s 2s/step - accuracy: 0.9619 - loss: 0.1579 - val_accuracy: 0.9280 - val_loss: 0.2495
Epoch 4/20
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9704 - loss: 0.1082Validation Accuracy at epoch 4: 0.9340000152587891
157/157 ━━━━━━━━━━━━━━━━━━━━ 368s 2s/step - accuracy: 0.9704 - loss: 0.1082 - val_accuracy: 0.9340 - val_loss: 0.3640
Epoch 5/20
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9785 - loss: 0.0710Validation Accuracy at epoch 5: 0.9179999828338623
157/157 ━━━━━━━━━━━━━━━━━━━━ 368s 2s/step - accuracy: 0.9785 - loss: 0.0710 - val_accuracy: 0.9180 - val_loss: 0.4304
Epoch 6/20
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9837 - loss: 0.0662Validation Accuracy at epoch 6: 0.8999999761581421
157/157 ━━━━━━━━━━━━━━━━━━━━ 368s 2s/step - accuracy: 0.9837 - loss: 0.0661 - val_accuracy: 0.9000 - val_loss: 0.5740
Epoch 7/20
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9747 - loss: 0.1187Validation Accuracy at epoch 7: 0.9169999957084656
157/157 ━━━━━━━━━━━━━━━━━━━━ 364s 2s/step - accuracy: 0.9747 - loss: 0.1186 - val_accuracy: 0.9170 - val_loss: 0.4680
Epoch 8/20
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9897 - loss: 0.0398Validation Accuracy at epoch 8: 0.9179999828338623
157/157 ━━━━━━━━━━━━━━━━━━━━ 377s 2s/step - accuracy: 0.9897 - loss: 0.0398 - val_accuracy: 0.9180 - val_loss: 0.4426
Epoch 9/20
...
157/157 ━━━━━━━━━━━━━━━━━━━━ 365s 2s/step - accuracy: 0.9996 - loss: 0.0036 - val_accuracy: 0.9160 - val_loss: 0.7517
Epoch 20/20
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 1.0000 - loss: 0.0011Validation Accuracy at epoch 20: 0.9129999876022339
157/157 ━━━━━━━━━━━━━━━━━━━━ 372s 2s/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 0.9130 - val_loss: 0.8579
```

2.2 Methodology – Acne Severity Assesment

2.2.1 Data Collection

For the development of an automated acne severity detection system, a comprehensive dataset was compiled from publicly available sources, including DermNet and Kaggle, ensuring a diverse representation of acne presentations across different skin types and severity levels. To maintain clinical accuracy and reliability, the collected images underwent a rigorous validation process supervised by two board-certified dermatologists. These experts meticulously reviewed each image, confirming proper classification into four distinct severity categories:

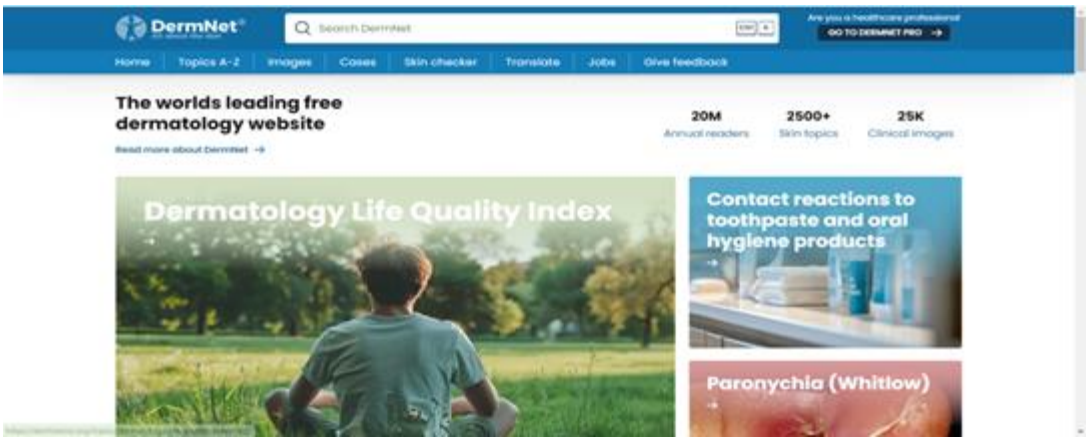
Level 0 (Mild Acne)

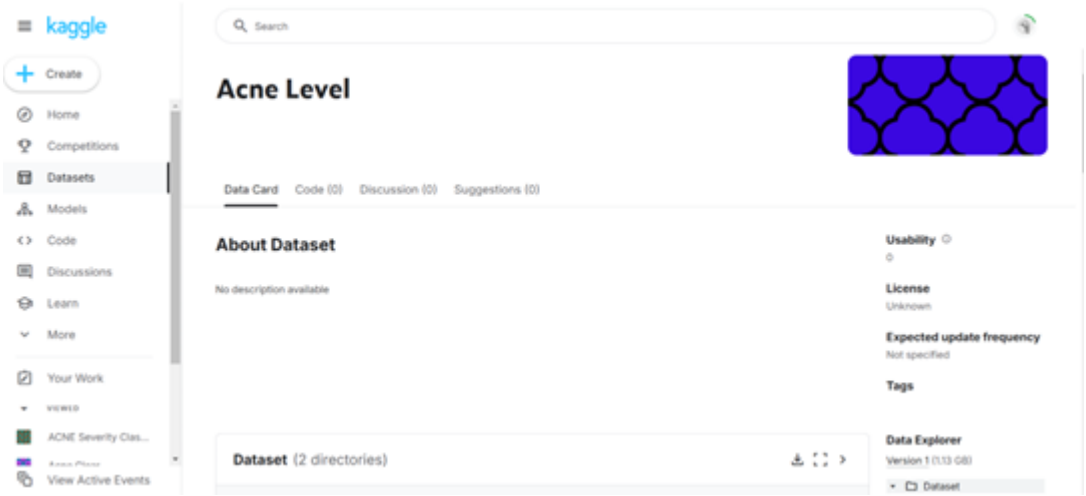
Level 1 (Moderate acne)

Level 2 (Severe acne)

Level 3 (High Severe acne)

The dermatologists' validation ensured that the dataset adhered to clinical standards, eliminating mislabeled or ambiguous cases that could compromise model performance. This structured and expert-verified dataset served as the foundation for training and evaluating deep learning models, enhancing the reliability of the automated severity assessment system.


DermNet
Home
Topics A-Z
Images
Cases
Skin checker
Translate
Jobs
Give feedback
The worlds leading free dermatology website
20M
2500+
25K
Dermatology Life Quality Index
Contact reactions to toothpaste and oral hygiene products
Paronychia (Whitlow)


kaggle
Create
Home
Competitions
Datasets
Models
Code
Discussions
Learn
More
Your Work
View Active Events
Search
Acne Level
Data Card
Code (0)
Discussion (0)
Suggestions (0)
About Dataset
No description available
Usability
License
Unknown
Expected update frequency
Not specified
Tags
Dataset (2 directories)
Data Explorer

October 30, 2024

Confirmation of Image Data Validation for Acne, Psoriasis, and Eczema

To Whom It May Concern,

We are pleased to formally confirm the successful completion of the image data validation process for acne, psoriasis, and eczema.

Our external supervisors, Dr. Saman Gunasekara and Dr. J.A.K.N Jayasinghe have conducted a thorough review and validation of the images. They have ensured that the images meet the required clinical standards of quality.

This validated dataset will serve as a crucial resource for our ongoing research endeavors aimed at developing diagnostic tools for these skin conditions.

Sincerely,

.....

Gimmana M.R.M
IT21176456
Research Project ID : 24-25J-210

Approved by:

.....

Dr. Saman Gunasekara
MBBS,MD.(Dermatology)
Consultant Dermatologist
Teaching Hospital – Kalubowila.

.....

Dr. J.A.K.N Jayasinghe
MBBS, MD (Dermatology)
Senior Registrar in Dermatology
Teaching Hospital – Kalubowila.

2.2.2 Data Preprocessing

In the data preprocessing phase of this research, significant steps were taken to prepare the acne severity dataset for effective model training and analysis. The preprocessing pipeline was carefully designed to handle the unique characteristics of dermatological images while ensuring the data quality and consistency needed for accurate classification and lesion detection.

The process began with loading and organizing the dataset, which was structured into four distinct severity categories (Level 0 to Level 3). Each image underwent resizing to a standardized dimension of 224×224 pixels, a crucial step for maintaining uniformity across the dataset and ensuring compatibility with the subsequent deep learning architecture. The images were then converted from BGR to RGB color space, addressing potential inconsistencies in color representation that could affect the model's ability to accurately identify acne lesions.

A critical aspect of the preprocessing involved data augmentation, implemented to enhance the model's robustness and generalization capabilities. The augmentation pipeline included random rotations (up to 20 degrees), width and height shifts (20% of total dimensions), shear transformations (20% intensity), zoom variations (20% range), and horizontal flips. These transformations effectively expanded the training dataset artificially, helping the model learn invariant features and reducing the risk of overfitting to specific image orientations or compositions.

For the lesion detection component, specialized preprocessing was implemented to identify and highlight red areas in the images, corresponding to inflamed acne lesions. This involved converting images to HSV color space and applying carefully tuned color thresholds to isolate red regions. Morphological operations including opening and closing

were employed to refine the detected areas, removing noise and filling gaps in the lesion masks. The system then generated bounding boxes around these detected regions, providing valuable spatial information about lesion locations.

The preprocessing pipeline also incorporated comprehensive data splitting, dividing the dataset into training and validation subsets while maintaining the original class distribution through stratified sampling. This approach ensured that the model evaluation would be representative of real-world performance across all severity levels. Throughout the process, normalization was consistently applied, scaling pixel values to a 0-1 range to facilitate stable and efficient model training.

To support reproducibility and troubleshooting, the implementation included logging mechanisms that tracked all preprocessing operations, recording details such as image paths, processing times, and any encountered errors. This systematic approach to data preparation created a solid foundation for the subsequent model development phase, ensuring that the training data was both representative of the problem domain and optimized for deep learning applications. The careful attention to preprocessing details contributed significantly to the model's ability to learn meaningful patterns and achieve accurate acne severity classification and lesion detection.

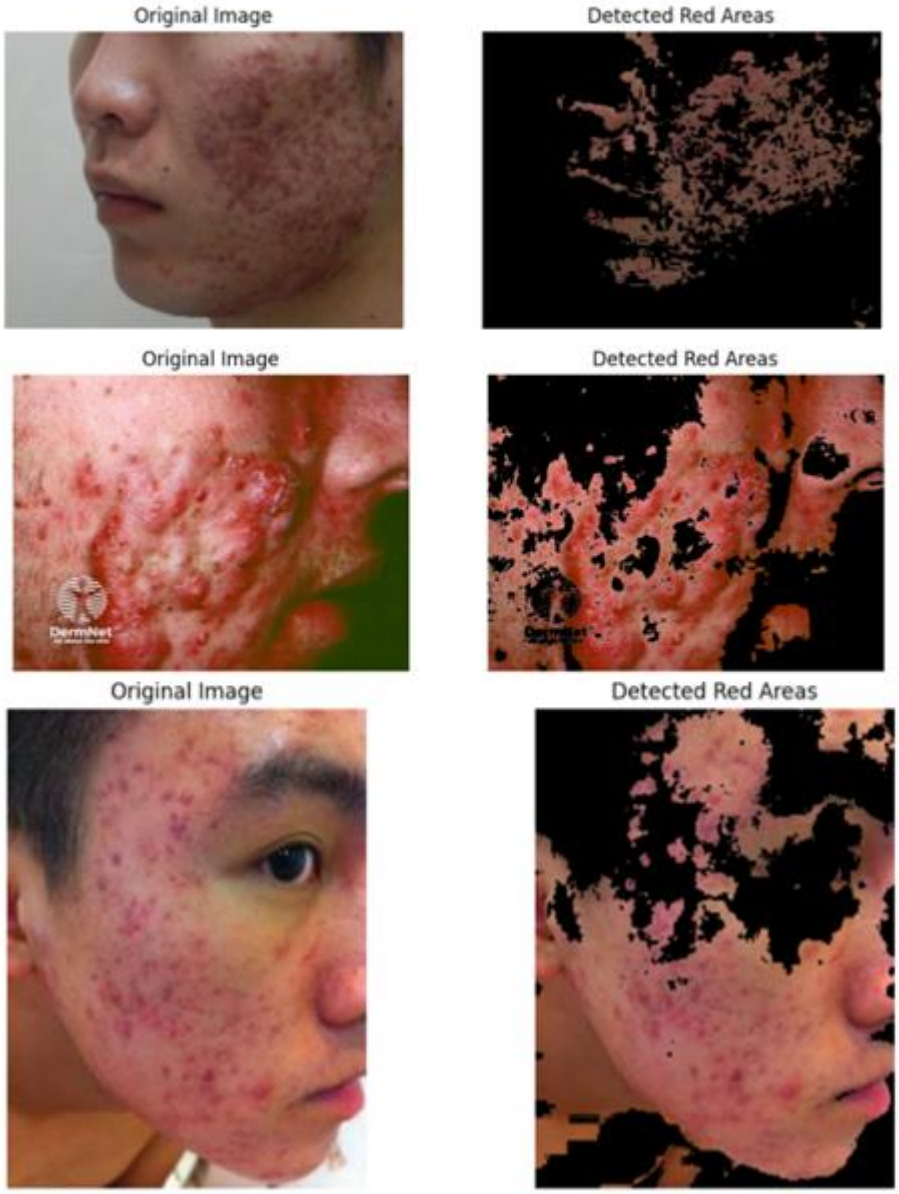


Image 6 : Acne Lesion Laclization using OpenCV

2.2.3 Model Building and Training

Five deep learning architectures, VGG16, CNN, ResNet50V2, EfficientNetB0, Siamese network with CNN were rigorously evaluated for the acne severity detection task, with the optimal model being selected based on comparative performance analysis.

Approach 1 : EfficientNetB0

```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import EfficientNetB0
from sklearn.model_selection import train_test_split
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

```python
# Data Directory and Categories
data_dir = '../New_Severity_Dataset'
categories = ['Level 0', 'Level 1', 'Level 2', 'Level 3']
img_height, img_width = 224, 224
batch_size = 16
```

```python
# Function to Load Images and Labels
def load_data(data_dir, categories):
    images, labels = [], []
    for category in categories:
        category_path = os.path.join(data_dir, category)
        for img_name in os.listdir(category_path):
            img_path = os.path.join(category_path, img_name)
            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = cv2.resize(img, (img_width, img_height))
            images.append(img)
            labels.append(categories.index(category))
    return np.array(images), np.array(labels)
```

```python
import os

data_dir = '../New_Severity_Dataset'
print("Does data directory exist?", os.path.exists(data_dir))

for category in categories:
    category_path = os.path.join(data_dir, category)
    print(f"Checking: {category_path} -> Exists? {os.path.exists(category_path)}")
```

[5]

```
Does data directory exist? True
Checking: ../New_Severity_Dataset\Level 0 -> Exists? True
Checking: ../New_Severity_Dataset\Level 1 -> Exists? True
Checking: ../New_Severity_Dataset\Level 2 -> Exists? True
Checking: ../New_Severity_Dataset\Level 3 -> Exists? True
```

```python
# Load and Normalize Data
images, labels = load_data(data_dir, categories)
images = images / 255.0
```

[6]

```python
# Train-Test Split
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42, stratify=labels)
```

```python
# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    fill_mode='nearest'
)

datagen.fit(X_train)
```

+ Code + Markdown

```python
# EfficientNetB0 Model with Additional Layers
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
base_model.trainable = True  # Enable fine-tuning
```

```python
# EfficientNetB0 Model with Additional Layers
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
base_model.trainable = True  # Enable fine-tuning
```

```python
model1 = Sequential([
    base_model,
    BatchNormalization(),
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(4, activation='softmax')
])
```

```python
# Compile Model with Optimized Learning Rate
model1.compile(optimizer=Adam(learning_rate=0.00005), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```python
# Train Model with Early Stopping
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)

history = model1.fit(datagen.flow(X_train, y_train, batch_size=batch_size),
                     validation_data=(X_val, y_val),
                     epochs=50,
                     verbose=1)
```

```
152/152 ━━━━━━━━━━━━━━━━━━━━ 325s 2s/step - accuracy: 0.3743 - loss: 1.2458 - val_accuracy: 0.2033 - val_loss: 1.7409
Epoch 3/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 340s 2s/step - accuracy: 0.4500 - loss: 1.1257 - val_accuracy: 0.2099 - val_loss: 1.7519
Epoch 4/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 334s 2s/step - accuracy: 0.4790 - loss: 1.0355 - val_accuracy: 0.2066 - val_loss: 2.0215
Epoch 5/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 346s 2s/step - accuracy: 0.5351 - loss: 0.9940 - val_accuracy: 0.3752 - val_loss: 1.2509
Epoch 6/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 348s 2s/step - accuracy: 0.5586 - loss: 0.9422 - val_accuracy: 0.4397 - val_loss: 1.1992
Epoch 7/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 338s 2s/step - accuracy: 0.5661 - loss: 0.9104 - val_accuracy: 0.5058 - val_loss: 1.0958
Epoch 8/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 338s 2s/step - accuracy: 0.5568 - loss: 0.9005 - val_accuracy: 0.4182 - val_loss: 1.2104
Epoch 9/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 371s 2s/step - accuracy: 0.6059 - loss: 0.8382 - val_accuracy: 0.2364 - val_loss: 1.6063
Epoch 10/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 372s 2s/step - accuracy: 0.6159 - loss: 0.8252 - val_accuracy: 0.2149 - val_loss: 1.8371
Epoch 11/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 371s 2s/step - accuracy: 0.6319 - loss: 0.7852 - val_accuracy: 0.2860 - val_loss: 1.4170
Epoch 12/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 371s 2s/step - accuracy: 0.6288 - loss: 0.7897 - val_accuracy: 0.2595 - val_loss: 13.7978
Epoch 13/50
...
Epoch 49/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 607s 4s/step - accuracy: 0.7939 - loss: 0.4880 - val_accuracy: 0.2397 - val_loss: 7852.8511
Epoch 50/50
152/152 ━━━━━━━━━━━━━━━━━━━━ 443s 3s/step - accuracy: 0.7996 - loss: 0.5050 - val_accuracy: 0.3835 - val_loss: 2.5663
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Approach 2 : CNN

```python
import numpy as np
from PIL import Image
from glob import glob
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
import os
import cv2

# Initialize empty lists to store images and labels
X = []
Y = []
```

```python
# Directories for each face direction (bottom, top, left, right)
directions = ['Level 0', 'Level 1', 'Level 2', 'Level 3']
image_directory = '../New_Severity_Dataset'  # Replace with the actual path
img_height, img_width = 224, 224  # Resizing dimensions
batch_size = 16
# Function to load images and check counts
def load_data(data_dir, categories):
    images, labels = [], []
    class_counts = {category: 0 for category in categories}  # Track per-class count
    for category in categories:
        category_path = os.path.join(data_dir, category)
        if not os.path.exists(category_path):
            print(f"⚠️ Warning: Folder {category_path} does not exist!")
            continue
        for img_name in os.listdir(category_path):
            img_path = os.path.join(category_path, img_name)
            # Skip non-image files
            if not img_path.lower().endswith(('.png', '.jpg', '.jpeg')):
                continue
            img = cv2.imread(img_path)
            if img is None:
                print(f"❌ Error: Could not read {img_path}")
                continue
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = cv2.resize(img, (img_width, img_height))
            images.append(img)
            labels.append(categories.index(category))
```

```
            labels.append(categories.index(category))
            class_counts[category] += 1  # Update count
    print("✅ Class distribution:", class_counts)
    return np.array(images), np.array(labels)
```

```
# Load dataset
X, Y = load_data(image_directory, directions)
```

```
✅ Class distribution: {'Level 0': 714, 'Level 1': 906, 'Level 2': 783, 'Level 3': 621}
```

```
# One-hot encode labels
Y = to_categorical(Y, num_classes=len(directions))
```

```
# Check unique classes in the one-hot encoded labels
unique_classes = np.argmax(Y, axis=1)  # Convert one-hot to integer labels
class_counts = np.unique(unique_classes, return_counts=True)

# Display the unique classes and their counts
for label, count in zip(class_counts[0], class_counts[1]):
    print(f"Class {label}: {count} samples")
```

```
Class 0: 714 samples
Class 1: 906 samples
Class 2: 783 samples
Class 3: 621 samples
```

```
print("Dataset successfully loaded!")
print(f"X shape: {X.shape}, Y shape: {Y.shape}")  # Debugging output
```

```
Dataset successfully loaded!
X shape: (3024, 224, 224, 3), Y shape: (3024, 4)
```

```
# Split data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=42)
```

[34]

```
# Check unique classes in the one-hot encoded labels
unique_classes = np.argmax(y_train, axis=1)  # Convert one-hot to integer labels
class_counts = np.unique(unique_classes, return_counts=True)

# Display the unique classes and their counts
for label, count in zip(class_counts[0], class_counts[1]):
    print(f"Class {label}: {count} samples")
```

[35]

```
Class 0: 554 samples
Class 1: 730 samples
Class 2: 626 samples
Class 3: 509 samples
```

+ Code + Markdown

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
model4 = Sequential()
# First convolutional block
model4.add(Conv2D(32, (3, 3), padding='same', input_shape=(224, 224, 3), activation='relu'))
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.3))
# Second convolutional block
model4.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.3))
# Flatten layer
model4.add(Flatten())
# Fully connected layer
model4.add(Dense(64, activation='relu'))
model4.add(Dropout(0.4))  # Increased dropout to prevent overfitting
# Output layer (4 classes, softmax)
model4.add(Dense(4, activation='softmax'))
# Compile the model
model4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model4.summary()
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 112, 112, 32)	0
dropout_13 (Dropout)	(None, 112, 112, 32)	0
conv2d_7 (Conv2D)	(None, 112, 112, 64)	18,496
max_pooling2d_7 (MaxPooling2D)	(None, 56, 56, 64)	0
dropout_14 (Dropout)	(None, 56, 56, 64)	0
flatten_2 (Flatten)	(None, 200704)	0
dense_10 (Dense)	(None, 64)	12,845,120
dropout_15 (Dropout)	(None, 64)	0
dense_11 (Dense)	(None, 4)	260

```python
from tensorflow.keras.callbacks import ModelCheckpoint

# Define the checkpoint callback to save the best model during training
checkpoint = ModelCheckpoint(
    'face_direction_model.keras',  # Save the model as a .h5 file
    monitor='val_loss',  # Monitor validation loss
    save_best_only=True,  # Save only the best model
    mode='min',
    verbose=1
)
```

```python
from tensorflow.keras.callbacks import ModelCheckpoint

# Define the checkpoint callback to save the best model during training
checkpoint = ModelCheckpoint(
    'face_direction_model.keras',  # Save the model as a .h5 file
    monitor='val_loss',  # Monitor validation loss
    save_best_only=True,  # Save only the best model
    mode='min',
    verbose=1
)

# Train the model
history4 = model4.fit(
    x_train, y_train,  # Training data
    validation_data=(x_test, y_test),  # Validation data
    epochs=100,  # Number of epochs
    batch_size=32,  # Batch size
    callbacks=[checkpoint]  # Use the checkpoint callback
)
```

43]

```
76/76 ━━━━━━━━━━━━━━━━━━━━ 60s 793ms/step - accuracy: 0.8821 - loss: 0.3019 - val_accuracy: 0.7322 - val_loss: 2.1287
Epoch 2/100
76/76 ━━━━━━━━━━━━━━━━━━━━ 0s 703ms/step - accuracy: 0.8860 - loss: 0.2985
Epoch 2: val_loss did not improve from 2.12867
76/76 ━━━━━━━━━━━━━━━━━━━━ 57s 749ms/step - accuracy: 0.8860 - loss: 0.2984 - val_accuracy: 0.7289 - val_loss: 2.3274
Epoch 3/100
76/76 ━━━━━━━━━━━━━━━━━━━━ 0s 711ms/step - accuracy: 0.8793 - loss: 0.2828
Epoch 3: val_loss improved from 2.12867 to 1.96123, saving model to face_direction_model.keras
76/76 ━━━━━━━━━━━━━━━━━━━━ 60s 784ms/step - accuracy: 0.8793 - loss: 0.2831 - val_accuracy: 0.7306 - val_loss: 1.9612
Epoch 4/100
76/76 ━━━━━━━━━━━━━━━━━━━━ 0s 710ms/step - accuracy: 0.8729 - loss: 0.3144
Epoch 4: val_loss did not improve from 1.96123
76/76 ━━━━━━━━━━━━━━━━━━━━ 58s 757ms/step - accuracy: 0.8729 - loss: 0.3143 - val_accuracy: 0.7240 - val_loss: 2.4744
Epoch 5/100
76/76 ━━━━━━━━━━━━━━━━━━━━ 0s 704ms/step - accuracy: 0.8829 - loss: 0.2952
Epoch 5: val_loss did not improve from 1.96123
76/76 ━━━━━━━━━━━━━━━━━━━━ 57s 752ms/step - accuracy: 0.8829 - loss: 0.2952 - val_accuracy: 0.7256 - val_loss: 2.3634
Epoch 6/100
76/76 ━━━━━━━━━━━━━━━━━━━━ 0s 696ms/step - accuracy: 0.8795 - loss: 0.2967
Epoch 6: val_loss did not improve from 1.96123
76/76 ━━━━━━━━━━━━━━━━━━━━ 57s 743ms/step - accuracy: 0.8795 - loss: 0.2967 - val_accuracy: 0.7240 - val_loss: 2.2824
Epoch 7/100
...
Epoch 100/100
76/76 ━━━━━━━━━━━━━━━━━━━━ 0s 539ms/step - accuracy: 0.9095 - loss: 0.2225
Epoch 100: val_loss did not improve from 1.96123
76/76 ━━━━━━━━━━━━━━━━━━━━ 44s 574ms/step - accuracy: 0.9094 - loss: 0.2227 - val_accuracy: 0.7405 - val_loss: 2.6785
```

Approach 3 : VGG16

Changed the model architecture with VGG16 base model

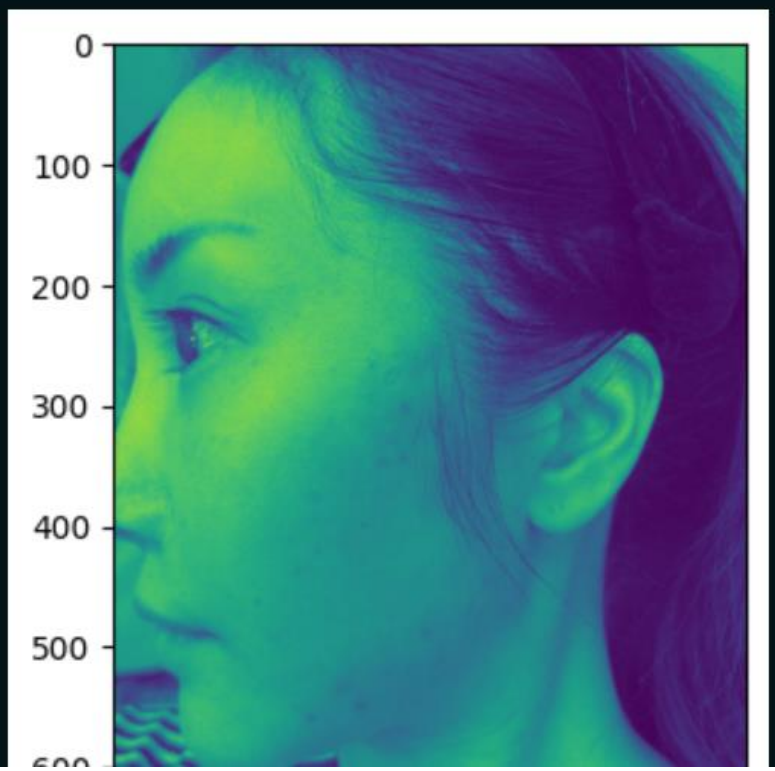
```
#Importing libraries
import cv2
import glob
import os
import matplotlib.pyplot as plt
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from pathlib import Path
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score , confusion_matrix
from sklearn import preprocessing

#tensorflow libraries
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models, Model, Sequential
from keras.layers import Dense, Dropout , BatchNormalization
from tensorflow.keras.callbacks import Callback, EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
```

[56]

```
# Define constants
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 16
EPOCHS = 20

# Load the DataFrame
path = "../New_Severity_Dataset"
data = {"imgpath": [], "labels": []}
category = os.listdir(path)
for folder in category:
    folderpath = os.path.join(path, folder)
    filelist = os.listdir(folderpath)
    for file in filelist:
        fpath = os.path.join(folderpath, file)
        data["imgpath"].append(fpath)
        data["labels"].append(folder)
df = pd.DataFrame(data)
```

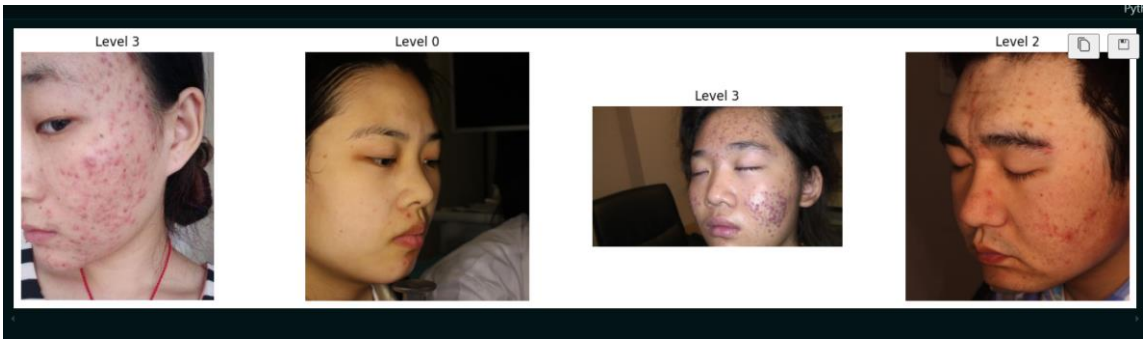
[57]

```
filepath = "../New_Severity_Dataset/Level 0"

for img_path in os.listdir(filepath):
    img = cv2.imread(os.path.join(filepath, img_path), 0)
    plt.imshow(img)
    plt.xlabel(filepath.split("/")[-1])
    plt.show()
    break
```



```python
# Split the DataFrame into train, validation, and test sets
train_df, temp_df = train_test_split(df, train_size=0.70, shuffle=True, random_state=124)
valid_df, test_df = train_test_split(temp_df, train_size=0.70, shuffle=True, random_state=124)
```

```python
#Defining size of displayed image
plt.figure(figsize=(20,20))

#Displaying images from test dataframe
for i, row in test_df.sample(n=4).reset_index().iterrows():
    plt.subplot(4,4,i+1)
    image_path = row['imgpath']
    image = Image.open(image_path)
    plt.imshow(image)
    plt.title(row["labels"])
    plt.axis('off')
plt.show()
```



```python
from sklearn.preprocessing import LabelEncoder

# Encode labels
label_encoder = LabelEncoder()
train_labels = label_encoder.fit_transform(train_df["labels"])
valid_labels = label_encoder.transform(valid_df["labels"])
test_labels = label_encoder.transform(test_df["labels"])
```

```python
from tensorflow.keras.utils import to_categorical

# Convert labels to one-hot encoded vectors
num_classes = len(set(train_labels))
train_labels = to_categorical(train_labels, num_classes=num_classes)
valid_labels = to_categorical(valid_labels, num_classes=num_classes)
test_labels = to_categorical(test_labels, num_classes=num_classes)
```

```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Define function to load and preprocess images
def load_images(image_paths):
    images = []
    for img_path in image_paths:
        img = load_img(img_path, target_size=IMAGE_SIZE)
        img = img_to_array(img)
        img = img / 255.0  # Normalize pixel values
        images.append(img)
    return np.array(images)

# Load and preprocess train, validation, and test images
train_images = load_images(train_df["imgpath"])
valid_images = load_images(valid_df["imgpath"])
test_images = load_images(test_df["imgpath"])
```

+ Code + Markdown

```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# Define custom callback to stop training once the accuracy reaches 0.9
class MyCallback(Callback):
    def on_epoch_end(self, epoch, logs=None):
        val_accuracy = logs.get("val_accuracy")
        print(f"Validation Accuracy at epoch {epoch + 1}: {val_accuracy}")
        if val_accuracy >= 0.9:
            print("\nReached 90% accuracy! Stopping training...")
            self.model.stop_training = True

# Load the pre-trained VGG16 model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3))

# Freeze the convolutional base
base_model.trainable = False

# Build the model
model5 = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation="relu"),
    Dense(num_classes, activation="softmax")
])
```

```python
# Compile the model
model5.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

```python
model5.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
flatten_3 (Flatten)	(None, 25088)	0
dense_12 (Dense)	(None, 256)	6,422,784
dense_13 (Dense)	(None, 4)	1,028

```python
# Train the model with the custom callback
history5 = model5.fit(
    x=train_images, y=train_labels,
    validation_data=(valid_images, valid_labels),
    batch_size=BATCH_SIZE, epochs=30,
    callbacks=[MyCallback()]
)
```

```
133/133 ━━━━━━━━━━━━━━━━━━━━ 483s 3s/step - accuracy: 0.5952 - loss: 2.1045 - val_accuracy: 0.5811 - val_loss: 0.9557
Epoch 2/30
133/133 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.6722 - loss: 0.7699Validation Accuracy at epoch 2: 0.6551181077957153
133/133 ━━━━━━━━━━━━━━━━━━━━ 426s 3s/step - accuracy: 0.6721 - loss: 0.7700 - val_accuracy: 0.6551 - val_loss: 0.8597
Epoch 3/30
133/133 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.7609 - loss: 0.6140Validation Accuracy at epoch 3: 0.6519684791564941
133/133 ━━━━━━━━━━━━━━━━━━━━ 438s 3s/step - accuracy: 0.7609 - loss: 0.6140 - val_accuracy: 0.6520 - val_loss: 0.8970
Epoch 4/30
133/133 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.7841 - loss: 0.5455Validation Accuracy at epoch 4: 0.6299212574958801
133/133 ━━━━━━━━━━━━━━━━━━━━ 465s 3s/step - accuracy: 0.7842 - loss: 0.5455 - val_accuracy: 0.6299 - val_loss: 0.8514
Epoch 5/30
133/133 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8365 - loss: 0.4377Validation Accuracy at epoch 5: 0.6582677364349365
133/133 ━━━━━━━━━━━━━━━━━━━━ 443s 3s/step - accuracy: 0.8364 - loss: 0.4379 - val_accuracy: 0.6583 - val_loss: 0.8268
Epoch 6/30
133/133 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8413 - loss: 0.4179Validation Accuracy at epoch 6: 0.7181102633476257
133/133 ━━━━━━━━━━━━━━━━━━━━ 447s 3s/step - accuracy: 0.8412 - loss: 0.4180 - val_accuracy: 0.7181 - val_loss: 0.8352
Epoch 7/30
133/133 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8582 - loss: 0.3610Validation Accuracy at epoch 7: 0.7370078563690186
133/133 ━━━━━━━━━━━━━━━━━━━━ 448s 3s/step - accuracy: 0.8582 - loss: 0.3612 - val_accuracy: 0.7370 - val_loss: 0.8038
Epoch 8/30
133/133 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8614 - loss: 0.3632Validation Accuracy at epoch 8: 0.7165354490280151
133/133 ━━━━━━━━━━━━━━━━━━━━ 446s 3s/step - accuracy: 0.8614 - loss: 0.3632 - val_accuracy: 0.7165 - val_loss: 0.9236
Epoch 9/30
...
133/133 ━━━━━━━━━━━━━━━━━━━━ 454s 3s/step - accuracy: 0.8746 - loss: 0.2962 - val_accuracy: 0.7433 - val_loss: 1.1038
Epoch 30/30
133/133 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8953 - loss: 0.2698Validation Accuracy at epoch 30: 0.7338582873344421
133/133 ━━━━━━━━━━━━━━━━━━━━ 467s 4s/step - accuracy: 0.8953 - loss: 0.2698 - val_accuracy: 0.7339 - val_loss: 0.9472
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Approach 4 : ResNet50V2

```python
import cv2
import glob
import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from pathlib import Path
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score, confusion_matrix
from sklearn import preprocessing

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models, Model, Sequential
from keras.layers import Dense, Dropout, BatchNormalization, Conv2D, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.callbacks import Callback, EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.applications import VGG16, ResNet50V2  # Import ResNet50V2

# Constants (adjust as needed)
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 16
EPOCHS = 30  # Increased epochs
NUM_CLASSES = 4 # Explicitly define number of classes
```

```python
# Load the DataFrame
path = "../New_Severity_Dataset"  # Replace with your actual path
data = {"imgpath": [], "labels": []}
category = os.listdir(path)
for folder in category:
    folderpath = os.path.join(path, folder)
    filelist = os.listdir(folderpath)
    for file in filelist:
        fpath = os.path.join(folderpath, file)
        data["imgpath"].append(fpath)
        data["labels"].append(folder)
df = pd.DataFrame(data)


# Split the DataFrame
train_df, temp_df = train_test_split(df, train_size=0.70, shuffle=True, random_state=124)
valid_df, test_df = train_test_split(temp_df, train_size=0.70, shuffle=True, random_state=124)


# Encode labels
label_encoder = LabelEncoder()
train_labels = label_encoder.fit_transform(train_df["labels"])
valid_labels = label_encoder.transform(valid_df["labels"])
test_labels = label_encoder.transform(test_df["labels"])
```

```python
# Convert labels to one-hot encoded vectors
train_labels = tf.keras.utils.to_categorical(train_labels, num_classes=num_classes)
valid_labels = tf.keras.utils.to_categorical(valid_labels, num_classes=num_classes)
test_labels = tf.keras.utils.to_categorical(test_labels, num_classes=num_classes)


# Load and preprocess images
def load_images(image_paths):
    images = []
    for img_path in image_paths:
        img = load_img(img_path, target_size=IMAGE_SIZE)
        img = img_to_array(img)
        images.append(img)
    return np.array(images)

train_images = load_images(train_df["imgpath"])
valid_images = load_images(valid_df["imgpath"])
test_images = load_images(test_df["imgpath"])
```

```python
# Data Augmentation (Important for improving model generalization)
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

valid_datagen = ImageDataGenerator(rescale=1./255)  # Only rescaling for validation

train_generator = train_datagen.flow(train_images, train_labels, batch_size=BATCH_SIZE)
validation_generator = valid_datagen.flow(valid_images, valid_labels, batch_size=BATCH_SIZE)

# Model Building (ResNet50V2 with some modifications)
base_model = ResNet50V2(weights="imagenet", include_top=False, input_shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3))
base_model.trainable = False  # Start with freezing, fine-tune later
```

```python
model8 = Sequential()
model8.add(base_model)
model8.add(GlobalAveragePooling2D())  # Use GAP for dimensionality reduction
model8.add(BatchNormalization()) # Added Batch Normalization
model8.add(Dense(512, activation='relu')) # Increased Dense layer size
model8.add(Dropout(0.5))  # Added Dropout
model8.add(Dense(NUM_CLASSES, activation='softmax'))

# Compile the model
model8.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Callbacks (Improved)
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True) # More patient
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=0.00001) # Reduce learning rate
model_checkpoint = ModelCheckpoint("acne_sever_VGG1.keras", monitor='val_accuracy', save_best_only=True) # Save best model

# Train the model with generators and improved callbacks
history = model8.fit(
    train_generator,
    steps_per_epoch=len(train_images) // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=len(valid_images) // BATCH_SIZE,
    callbacks=[early_stopping, reduce_lr, model_checkpoint]
)
```

```
132/132 ━━━━━━━━━━━━━━━━━━━━ 139s 1s/step - accuracy: 0.5092 - loss: 1.4460 - val_accuracy: 0.5240 - val_loss: 1.0482 - learning_rate: 0.0010
Epoch 4/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 31s 229ms/step - accuracy: 0.4375 - loss: 1.0479 - val_accuracy: 0.5128 - val_loss: 1.0424 - learning_rate: 0.0010
Epoch 5/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 140s 1s/step - accuracy: 0.5386 - loss: 1.1579 - val_accuracy: 0.5064 - val_loss: 1.0164 - learning_rate: 0.0010
Epoch 6/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 31s 232ms/step - accuracy: 0.3125 - loss: 1.1671 - val_accuracy: 0.5160 - val_loss: 1.0160 - learning_rate: 0.0010
Epoch 7/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 141s 1s/step - accuracy: 0.5822 - loss: 1.0232 - val_accuracy: 0.6058 - val_loss: 0.8702 - learning_rate: 0.0010
Epoch 8/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 33s 244ms/step - accuracy: 0.3125 - loss: 1.1215 - val_accuracy: 0.6090 - val_loss: 0.8681 - learning_rate: 0.0010
Epoch 9/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 140s 1s/step - accuracy: 0.5694 - loss: 0.9789 - val_accuracy: 0.6138 - val_loss: 0.9052 - learning_rate: 0.0010
Epoch 10/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 31s 234ms/step - accuracy: 0.6875 - loss: 0.8677 - val_accuracy: 0.6138 - val_loss: 0.9008 - learning_rate: 0.0010
Epoch 11/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 136s 1s/step - accuracy: 0.5806 - loss: 0.8973 - val_accuracy: 0.6106 - val_loss: 0.8665 - learning_rate: 0.0010
Epoch 12/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 34s 252ms/step - accuracy: 0.6875 - loss: 0.8241 - val_accuracy: 0.6186 - val_loss: 0.8619 - learning_rate: 0.0010
Epoch 13/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 138s 1s/step - accuracy: 0.6098 - loss: 0.8904 - val_accuracy: 0.6298 - val_loss: 0.8362 - learning_rate: 0.0010
Epoch 14/30
132/132 ━━━━━━━━━━━━━━━━━━━━ 31s 229ms/step - accuracy: 0.8125 - loss: 0.6873 - val_accuracy: 0.6106 - val_loss: 0.8423 - learning_rate: 0.0010
...
132/132 ━━━━━━━━━━━━━━━━━━━━ 136s 1s/step - accuracy: 0.6441 - loss: 0.8162 - val_accuracy: 0.6410 - val_loss: 0.8102 - learning_rate: 5.0000e-04
9/9 ━━━━━━━━━━━━━━━━━━━━ 22s 1s/step - accuracy: 0.2318 - loss: 1261.9784
Test Loss: 1253.623046875
Test Accuracy: 0.24542124569416046
```

Approach 5 : Siamese with CNN

```python
# Define constants
IMAGE_SIZE = (128, 128)
BATCH_SIZE = 8
EPOCHS = 10
NUM_PAIRS = 500  # Number of pairs for Siamese network training

# Load the dataset
path = "../SiamsesS"
data = {"imgpath": [], "labels": []}
categories = os.listdir(path)
for folder in categories:
    folderpath = os.path.join(path, folder)
    filelist = os.listdir(folderpath)
    for file in filelist:
        fpath = os.path.join(folderpath, file)
        data["imgpath"].append(fpath)
        data["labels"].append(folder)
df = pd.DataFrame(data)
```

[2]

```python
# Split the dataset into train, validation, and test sets
train_df, temp_df = train_test_split(df, train_size=0.70, shuffle=True, random_state=124)
valid_df, test_df = train_test_split(temp_df, train_size=0.70, shuffle=True, random_state=124)
```

```python
# Encode labels
label_encoder = LabelEncoder()
train_labels = label_encoder.fit_transform(train_df["labels"])
valid_labels = label_encoder.transform(valid_df["labels"])
test_labels = label_encoder.transform(test_df["labels"])

# Convert labels to one-hot encoded vectors
num_classes = len(set(train_labels))
train_labels = to_categorical(train_labels, num_classes=num_classes)
valid_labels = to_categorical(valid_labels, num_classes=num_classes)
test_labels = to_categorical(test_labels, num_classes=num_classes)

# Function to load and preprocess images
def load_images(image_paths):
    images = []
    for img_path in image_paths:
        img = load_img(img_path, target_size=IMAGE_SIZE)
        img = img_to_array(img)
        img = img / 255.0  # Normalize pixel values
        images.append(img)
    return np.array(images)
```

```python
# Load and preprocess images
train_images = load_images(train_df["imgpath"])
valid_images = load_images(valid_df["imgpath"])
test_images = load_images(test_df["imgpath"])

# Convert labels back to integers for pairing
train_labels_int = np.argmax(train_labels, axis=1)
valid_labels_int = np.argmax(valid_labels, axis=1)
test_labels_int = np.argmax(test_labels, axis=1)
```

```python
# Function to create pairs of images and labels
def create_pairs(images, labels, num_pairs=500):
    pairs = []
    pair_labels = []

    num_classes = len(set(labels))
    class_indices = [np.where(labels == i)[0] for i in range(num_classes)]

    for _ in range(num_pairs):
        # Randomly select two images from the same class
        class_idx = random.randint(0, num_classes - 1)
        idx1, idx2 = random.sample(list(class_indices[class_idx]), 2)
        pairs.append([images[idx1], images[idx2]])
        pair_labels.append(1)  # Same class

        # Randomly select two images from different classes
        class_idx1, class_idx2 = random.sample(range(num_classes), 2)
        idx1 = random.choice(class_indices[class_idx1].tolist())
        idx2 = random.choice(class_indices[class_idx2].tolist())
        pairs.append([images[idx1], images[idx2]])
        pair_labels.append(0)  # Different classes

    return np.array(pairs), np.array(pair_labels)

# Create pairs for training, validation, and testing
train_pairs, train_pair_labels = create_pairs(train_images, train_labels_int, num_pairs=NUM_PAIRS)
valid_pairs, valid_pair_labels = create_pairs(valid_images, valid_labels_int, num_pairs=200)
test_pairs, test_pair_labels = create_pairs(test_images, test_labels_int, num_pairs=100)
```

```python
# Define the base network using a pre-trained ResNet50 model
def create_base_network(input_shape):
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
    x = base_model.output
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.5)(x)
    return Model(base_model.input, x)
```

```python
# Define the Siamese network
input_shape = (IMAGE_SIZE[0], IMAGE_SIZE[1], 3)
base_network = create_base_network(input_shape)

input_a = Input(shape=input_shape)
input_b = Input(shape=input_shape)

output_a = base_network(input_a)
output_b = base_network(input_b)

distance = layers.Lambda(lambda x: tf.abs(x[0] - x[1]))([output_a, output_b])
output = layers.Dense(1, activation='sigmoid')(distance)

siamese_model = Model([input_a, input_b], output)

# Compile the Siamese network
siamese_model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

# Callbacks for Siamese network training
siamese_callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ModelCheckpoint('best_siamese_model.keras', monitor='val_accuracy', save_best_only=True, mode='max'),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6),
    TensorBoard(log_dir='./logs/siamese')
]
```

```python
# Train the Siamese network
siamese_model.fit(
    [train_pairs[:, 0], train_pairs[:, 1]], train_pair_labels,
    validation_data=([valid_pairs[:, 0], valid_pairs[:, 1]], valid_pair_labels),
    batch_size=BATCH_SIZE, epochs=50, callbacks=siamese_callbacks
)
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 200s 2s/step - accuracy: 0.6261 - loss: 0.8049 - val_accuracy: 0.5350 - val_loss: 1.0467 - learning_rate: 1.0000e-06
Epoch 3/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 198s 2s/step - accuracy: 0.6583 - loss: 0.7447 - val_accuracy: 0.5450 - val_loss: 1.0536 - learning_rate: 1.0000e-06
Epoch 4/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 198s 2s/step - accuracy: 0.6239 - loss: 0.7548 - val_accuracy: 0.5400 - val_loss: 1.0429 - learning_rate: 1.0000e-06
Epoch 5/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 205s 2s/step - accuracy: 0.6636 - loss: 0.7169 - val_accuracy: 0.5350 - val_loss: 1.0481 - learning_rate: 1.0000e-06
Epoch 6/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 211s 2s/step - accuracy: 0.6543 - loss: 0.7371 - val_accuracy: 0.5325 - val_loss: 1.0357 - learning_rate: 1.0000e-06
Epoch 7/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 214s 2s/step - accuracy: 0.6505 - loss: 0.7194 - val_accuracy: 0.5425 - val_loss: 1.0560 - learning_rate: 1.0000e-06
Epoch 8/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 209s 2s/step - accuracy: 0.6039 - loss: 0.8113 - val_accuracy: 0.5350 - val_loss: 1.0672 - learning_rate: 1.0000e-06
Epoch 9/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 209s 2s/step - accuracy: 0.6700 - loss: 0.6973 - val_accuracy: 0.5425 - val_loss: 1.0638 - learning_rate: 1.0000e-06
Epoch 10/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 206s 2s/step - accuracy: 0.6609 - loss: 0.7035 - val_accuracy: 0.5400 - val_loss: 1.0691 - learning_rate: 1.0000e-06
Epoch 11/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 206s 2s/step - accuracy: 0.6533 - loss: 0.6858 - val_accuracy: 0.5375 - val_loss: 1.0608 - learning_rate: 1.0000e-06
Epoch 12/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 204s 2s/step - accuracy: 0.6367 - loss: 0.7457 - val_accuracy: 0.5375 - val_loss: 1.0614 - learning_rate: 1.0000e-06
Epoch 13/50
...
Epoch 15/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 208s 2s/step - accuracy: 0.6757 - loss: 0.6834 - val_accuracy: 0.5375 - val_loss: 1.0671 - learning_rate: 1.0000e-06
Epoch 16/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 209s 2s/step - accuracy: 0.6591 - loss: 0.7039 - val_accuracy: 0.5375 - val_loss: 1.0730 - learning_rate: 1.0000e-06
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```python
# Save the base network (feature extractor)
base_network.save("siamese_feature_extractor.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This f
```

Standard CNN Model for Severity Detection

```python
# Load the base network (feature extractor)
base_network = tf.keras.models.load_model("siamese_feature_extractor.h5")

# Freeze the base network (optional)
base_network.trainable = False
```

```
WARNING:absl:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
```

```python
# Define the multi-class classification model
def create_multi_class_model(base_network, input_shape, num_classes):
    input = Input(shape=input_shape)
    features = base_network(input)
    output = layers.Dense(num_classes, activation='softmax')(features)
    return Model(input, output)

# Create the multi-class classification model
multi_class_model = create_multi_class_model(base_network, input_shape, num_classes)

# Compile the model
multi_class_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```python
# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_multi_class_model.keras', monitor='val_accuracy', save_best_only=True, mode='max')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6)
tensorboard = TensorBoard(log_dir='./logs', histogram_freq=1)
```

+ Code + Markdown

```python
# Callbacks for multi-class classification training
multi_class_callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ModelCheckpoint('best_multi_class_model.keras', monitor='val_accuracy', save_best_only=True, mode='max'),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6),
    TensorBoard(log_dir='./logs/multi_class')
]
```

+ Code + Markdown

```python
# Train the multi-class classification model
history = multi_class_model.fit(
    train_images, train_labels,
    validation_data=(valid_images, valid_labels),
    batch_size=BATCH_SIZE, epochs=100, callbacks=multi_class_callbacks
)
```

```
Epoch 3/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 46s 259ms/step - accuracy: 0.3113 - loss: 2.3494 - val_accuracy: 0.3129 - val_loss: 1.8458 - learning_rate: 1.0000e-04
Epoch 4/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 46s 257ms/step - accuracy: 0.3027 - loss: 2.3896 - val_accuracy: 0.3247 - val_loss: 1.7411 - learning_rate: 1.0000e-04
Epoch 5/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 45s 257ms/step - accuracy: 0.2936 - loss: 2.2437 - val_accuracy: 0.3412 - val_loss: 1.6571 - learning_rate: 1.0000e-04
Epoch 6/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 44s 249ms/step - accuracy: 0.3506 - loss: 2.2690 - val_accuracy: 0.3412 - val_loss: 1.5887 - learning_rate: 1.0000e-04
Epoch 7/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 46s 263ms/step - accuracy: 0.3456 - loss: 1.9862 - val_accuracy: 0.3576 - val_loss: 1.5464 - learning_rate: 1.0000e-04
Epoch 8/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 46s 257ms/step - accuracy: 0.3561 - loss: 1.8888 - val_accuracy: 0.3671 - val_loss: 1.5072 - learning_rate: 1.0000e-04
Epoch 9/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 45s 256ms/step - accuracy: 0.3765 - loss: 1.7876 - val_accuracy: 0.3765 - val_loss: 1.4685 - learning_rate: 1.0000e-04
Epoch 10/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 45s 255ms/step - accuracy: 0.3854 - loss: 1.8230 - val_accuracy: 0.3835 - val_loss: 1.4387 - learning_rate: 1.0000e-04
Epoch 11/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 44s 250ms/step - accuracy: 0.3672 - loss: 1.8172 - val_accuracy: 0.3812 - val_loss: 1.4241 - learning_rate: 1.0000e-04
Epoch 12/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 51s 290ms/step - accuracy: 0.3890 - loss: 1.7997 - val_accuracy: 0.3859 - val_loss: 1.3969 - learning_rate: 1.0000e-04
Epoch 13/100
...
Epoch 99/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 79s 446ms/step - accuracy: 0.5254 - loss: 0.9294 - val_accuracy: 0.4000 - val_loss: 1.0590 - learning_rate: 2.0000e-05
Epoch 100/100
177/177 ━━━━━━━━━━━━━━━━━━━━ 80s 449ms/step - accuracy: 0.5230 - loss: 0.9714 - val_accuracy: 0.4000 - val_loss: 1.0586 - learning_rate: 2.0000e-05
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

2.3 Commercialization Aspect of the Product

The proposed system, Automated Advanced Acne Lesion Analysis and Monitoring with Artificial Intelligence, offers a comprehensive solution that integrates acne detection, severity classification, and patient healing tracking. This system is designed not as a replacement for dermatologists but as a powerful clinical decision support tool that works in parallel with healthcare professionals. By incorporating real-time AI analysis and continuous progress monitoring, the system enhances diagnostic precision, streamlines workflows, and ultimately improves patient outcomes.

There is a significant commercial opportunity in both the healthcare and personal wellness sectors. Acne remains one of the most common skin conditions globally, affecting millions of individuals across all age groups. Currently, diagnosis and tracking rely heavily on visual assessment, which can be subjective and inconsistent. This system addresses those limitations by providing standardized, AI-driven evaluation, enabling dermatologists to assess acne presence and severity with high accuracy and consistency. With accuracy rates exceeding 99% for detection and over 70% for severity classification, the system shows strong potential for adoption in dermatology practices, telemedicine platforms, and even hospital outpatient departments.

A unique strength of this solution is its integrated healing tracking module, which allows dermatologists to monitor patient progress over time. This longitudinal view, supported by periodic image analysis and automated comparison, can significantly reduce follow-up assessment time while providing objective measurements of treatment effectiveness. From a commercial standpoint, this feature can be a key differentiator when pitching to clinics, hospitals, and skincare centers, offering not just diagnosis but treatment monitoring and reporting capabilities.

Additionally, the system has been developed with dermatologist involvement at every stage, including data collection, annotation, and validation. This collaboration ensures medical credibility, fosters trust in clinical environments, and paves the way for smoother regulatory approval processes, such as CE marking or FDA clearance, where needed. As such, it can be positioned as a clinically validated AI assistant, rather than a black-box consumer tool.

Commercial deployment can take multiple forms. In private dermatology clinics, the system can be installed as a consultation assistant and used to enhance clinical documentation and treatment planning. In teledermatology settings, it can serve as a pre-screening and follow-up tool, enabling remote assessment without compromising on accuracy. For enterprise B2B partnerships, the system can be licensed as a white-labeled API, allowing integration into hospital management systems or digital health platforms.

Furthermore, its healing tracking module can support subscription-based services for long-term acne management plans. Patients could receive personalized reports and visual progress charts, while dermatologists benefit from efficient monitoring tools and patient engagement features. This opens up avenues for value-based care models, where outcomes are tracked and quantified over time.

In summary, the proposed AI-based acne analysis and monitoring system delivers a powerful blend of accuracy, automation, and clinical integration. With its dual role in enhancing diagnostic workflows and improving patient management, the system is well-positioned for commercialization across diverse healthcare landscapes. Its medically guided development, patient-centric features, and scalable design make it an ideal product for both clinical adoption and strategic healthcare partnerships, offering long-term value in the dermatology and digital health ecosystem.

2.4 Testing and Implementation

The development and evaluation of the proposed system were carried out in multiple phases to ensure its accuracy, reliability, and practical applicability in real-world dermatological workflows. The system comprises three core modules: acne detection, severity classification, and healing progress tracking, each of which underwent extensive testing using various machine learning models and evaluation strategies.

2.4.1 System Implementation

The system was implemented using Python and key deep learning frameworks including TensorFlow and Keras. For the acne detection module, pre-trained models such as VGG16, VGG19, ResNet50, EfficientNetB0, Siamese Networks, and a custom CNN architecture were explored. The severity classification module further extended this exploration with models like Vision Transformer (ViT), ResNet50V2, and custom CNNs. The system was trained and tested using a custom-curated dataset, with image-level annotations performed under the guidance of dermatologists to ensure clinical accuracy and relevance.

The user interface and patient management dashboard were designed for ease of use by dermatologists and included modules for patient image upload, automated analysis, severity labeling, and progress visualization over time. Healing progress tracking was implemented using time-series comparisons of patient images, where the system evaluated changes in acne regions and severity scores to visualize treatment efficacy.

2.4.2 Testing and Evaluation

Testing was conducted using a standard train-test split methodology, with 80% of the dataset allocated for training and 20% for validation and testing. All models were evaluated using key performance metrics including accuracy, error rate, confusion matrix analysis, precision, recall, and loss values.

For acne detection, the VGG16 model achieved the highest accuracy of 99.28%, with almost perfect classification results as evidenced by its confusion matrix. In severity

classification, the VGG16 and ViT models showed the best performance, with accuracy scores above 73%, and a relatively even distribution of correct predictions across all four severity levels.

Validation accuracy and loss graphs were plotted over epochs to monitor the learning dynamics of each model. Overfitting was mitigated using early stopping, dropout layers, and regularization techniques where applicable. The performance of each model was visualized using confusion matrices to provide insights into class-wise prediction strengths and misclassification patterns.

2.4.3 Real-World Validation

To ensure clinical relevance, testing was not limited to theoretical metrics. A select group of licensed dermatologists reviewed the system's predictions and provided feedback during both development and evaluation. Their insights were used to refine the severity classification labels, validate model predictions, and enhance the system's usability in clinical settings. This collaborative validation process significantly increased the trustworthiness and practical applicability of the solution.

2.4.4 Backend Implementation

The backend for the AI-powered dermatology system was developed using Flask, with a strong focus on modularity, scalability, and real-time image analysis. The application supports user authentication, disease detection, severity assessment, and dynamic image handling.

Import Statements

The system begins with importing essential libraries such as:

- Flask, request, jsonify, CORS – for handling API endpoints, requests, and enabling cross-origin communication.

- TensorFlow, PyTorch (torch), PIL, NumPy – to load and preprocess medical images for deep learning predictions.
- psycopg2.pool – to manage connections to the PostgreSQL database.
- dotenv and os – for secure access to environment variables.
- CloudinaryImageHandler (custom) – for uploading and retrieving skin images from the cloud.

.**Model Loading**

Deep Learning models are loaded at startup for different detection and classification tasks:

```
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

#eczema_detection_model = tf.keras.models.load_model('./models/eczema/Eczema_Detection_Model_with_Attention.h5', compile=False)
eczema_detection_model = tf.keras.models.load_model('./models/eczema/Eczema_Detection_Model_CNN.h5',compile=False)
acne_severity_model = tf.keras.models.load_model('./models/acne/acne_severity_detection_model.h5',compile=False)
```

Image 7 : Model Loading

Flask App and CORS Configuration

The Flask app is initialized with CORS settings allowing requests from both development and deployment frontend URLs. Environment variables are loaded using dotenv to configure database credentials securely.

```
app = Flask(__name__)

CORS(app, resources={
    r"/api/*": {
        "origins": ["http://localhost:5173", "http://localhost:4173"],
        "methods": ["GET", "POST", "PUT", "DELETE", "OPTIONS"],
        "allow_headers": ["Content-Type", "Authorization"]
    }
})
```

Image 8 : CORS of Flask App

Detection Endpoints

Each detection route handles image uploads, preprocessing, model inference, and result storage

```python
@app.route('/api/detect/acne', methods=['POST'])
def detect_acne():
    try:
        if 'image' not in request.files:
            return jsonify({'error': 'No image file provided'}), 400

        img_file = request.files['image']
        user_id = request.form.get('userId')
        body_part = request.form.get('bodyPart')

        if not user_id or not body_part:
            return jsonify({'error': 'User ID and body part are required'}), 400

        img = Image.open(img_file.stream)
        img = img.convert('RGB')
        img = img.resize((224, 224))
        img_array = np.array(img) / 255.0
        img_array = np.expand_dims(img_array, axis=0)

        detection_prediction = acne_detection_model.predict(img_array)
        print("Raw prediction:", detection_prediction)

        acne_probability = float(np.sum(detection_prediction[0][2:]))

        is_acne = acne_probability > 0.56

        print(f"Acne probability: {acne_probability}, Is Acne: {is_acne}")
```

Image 9 : Detection Endpoints

```python
def detect_acne():

        image_handler = CloudinaryImageHandler()
        image_url = image_handler.save_image(img_file, user_id)

        if not image_url:
            return jsonify({'error': 'Failed to save image'}), 500

        test_results = {
            'prediction': 'Acne' if is_acne else 'No Acne',
            'confidence': round(acne_probability * 100, 2),
            'detailed_scores': detection_prediction[0].tolist(),
            'body_part': body_part
        }

        # Save results to database
        conn = db_pool.getconn()
        try:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO skin_test_results
                (user_id, test_type, image_url, body_part, test_results)
                VALUES (%s, %s, %s, %s, %s)
                RETURNING id
                """,
                (user_id, 'acne', image_url, body_part,
                 json.dumps({
                     'detected': is_acne,
                     'confidence': round(acne_probability * 100, 2),
                     'raw_predictions': detection_prediction[0].tolist()
```

AcneDetection.tsx U app.py 2

backend > app.py > ...

```python
def detect_acne():

        # Save results to database
        conn = db_pool.getconn()
        try:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO skin_test_results
                (user_id, test_type, image_url, body_part, test_results)
                VALUES (%s, %s, %s, %s, %s)
                RETURNING id
                """,
                (user_id, 'acne', image_url, body_part,
                 json.dumps({
                     'detected': is_acne,
                     'confidence': round(acne_probability * 100, 2),
                     'raw_predictions': detection_prediction[0].tolist()
                 }))
            )
            result_id = cursor.fetchone()[0]
            conn.commit()
        except Exception as e:
            print(f"Database error: {str(e)}")
            raise
        finally:
            db_pool.putconn(conn)

        response_data = {
            'detected': is_acne,
            'confidence': round(acne_probability * 100, 2),
```

AcneDetection.tsx U app.py 2

backend > app.py > ...

```python
def detect_acne():
                'resultId': result_id
            }
            print("Response data:", response_data)

            return jsonify(response_data), 200

        except Exception as e:
            print(f"Error in detect_acne: {str(e)}")
            return jsonify({'error': str(e)}), 500
```

Severity Assessment

Each detection route handles image uploads, preprocessing, model inference, and result storage:

```python
@app.route('/api/assess/acne_severity', methods=['POST'])
def assess_acne_severity():
    try:
        if 'image' not in request.files:
            return jsonify({'error': 'No image file provided'}), 400

        img_file = request.files['image']
        result_id = request.form.get('resultId')  # Get the result ID from detection step

        if not result_id:
            return jsonify({'error': 'Result ID is required'}), 400

        # Process image for severity assessment
        img = Image.open(img_file.stream)
        img = img.convert('RGB')
        img = img.resize((224, 224))
        img_array = np.array(img) / 255.0
        img_array = np.expand_dims(img_array, axis=0)

        # Get severity prediction
        severity_prediction = acne_severity_model.predict(img_array)
        severity_scores = severity_prediction[0]
        severity_labels = ['Mild', 'Moderate', 'Severe', 'High Severe']

        severity_result = {
            'severity': {
                'level': severity_labels[np.argmax(severity_scores)],
                'confidence': round(float(np.max(severity_scores)) * 100, 2)
            },
            'severity_scores': {
```

Image 10 : Detection routs handling

AcneDetection.tsx U | app.py 2

backend > app.py > ...

```python
def assess_acne_severity():

        severity_result = {
            'severity': {
                'level': severity_labels[np.argmax(severity_scores)],
                'confidence': round(float(np.max(severity_scores)) * 100, 2)
            },
            'severity_scores': {
                label: round(float(score) * 100, 2)
                for label, score in zip(severity_labels, severity_scores)
            }
        }

        # Update the database record with severity results
        conn = db_pool.getconn()
        try:
            cursor = conn.cursor()
            cursor.execute("""
                UPDATE skin_test_results
                SET test_results = test_results || %s::jsonb
                WHERE id = %s
                """,
                (json.dumps(severity_result), result_id)
            )
            conn.commit()
        except Exception as e:
            print(f"Database error: {str(e)}")
            raise
        finally:
            db_pool.putconn(conn)
```

```python
def assess_acne_severity():

        return jsonify(severity_result), 200

    except Exception as e:
        print(f"Error in assess_acne_severity: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/api/detect/eczema', methods=['POST'])
def detect_eczema():
    try:
        if 'image' not in request.files:
            return jsonify({'error': 'No image file provided'}), 400

        img_file = request.files['image']
        user_id = request.form.get('userId')
        body_part = request.form.get('bodyPart')

        if not user_id or not body_part:
            return jsonify({'error': 'User ID and body part are required'}), 400

        img = Image.open(img_file.stream)
        img = img.convert('RGB')
        img = img.resize((224, 224))
        img_array = np.array(img) / 255.0
        img_array = np.expand_dims(img_array, axis=0)

        detection_prediction = eczema_detection_model.predict(img_array)
        print("Raw prediction:", detection_prediction)
```

Database Integration

The backend communicates with a PostgreSQL database using connection pooling. Each patient's test history is recorded, and corresponding skin analysis results are stored in the skin_test_results table. Queries are optimized for fetching both individual and aggregated patient data.

```python
        conn = db_pool.getconn()
        try:
            cursor = conn.cursor()
            cursor.execute("""
                UPDATE skin_test_results
                SET test_results = test_results || %s::jsonb
                WHERE id = %s
                """,
                (json.dumps(severity_result), result_id)
            )
            conn.commit()
        except Exception as e:
            print(f"Database error: {str(e)}")
            raise
        finally:
            db_pool.putconn(conn)

        return jsonify(severity_result), 200

    except Exception as e:
        print(f"Error in assess_psoriasis_severity: {str(e)}")
        return jsonify({'error': str(e)}), 500
```

Image 11 : Database Integration

Backend Testing

Backend testing was conducted using Postman, a widely adopted API testing tool, to verify the functionality and reliability of the system's endpoints. Various POST and GET requests were sent to test critical functionalities such as image upload, prediction generation, user authentication, and data retrieval. Each request was checked for expected inputs and outputs to ensure that the server responded correctly and consistently. Special attention was given to validating JSON responses, status codes, and error handling. This process ensured that the backend APIs performed as intended, supported smooth

integration with the frontend, and maintained secure and efficient communication with the database and machine learning models.

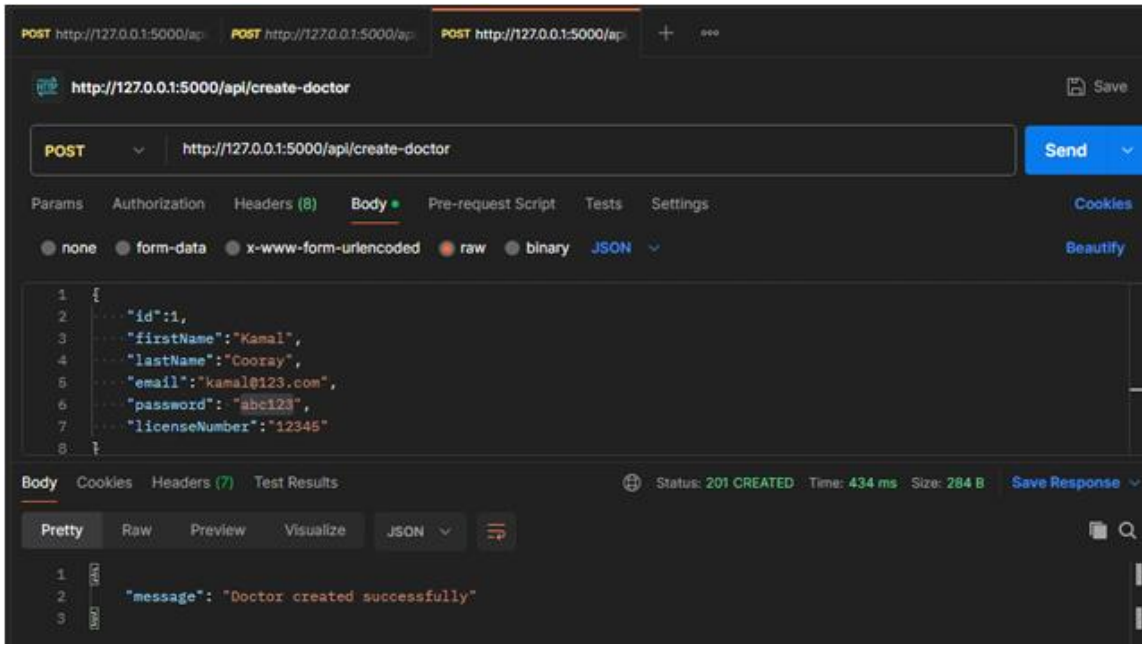


Image 12 : Testing using PostMan

2.4.5 Frontend Implementation

The frontend of the psoriasis detection and severity assessment system was developed using React, ensuring a responsive and user-friendly interface. Key features include image upload, body part selection, real-time result display, and dynamic visual feedback. The application integrates with backend APIs to provide accurate detection and severity classification while maintaining a smooth user experience.

Detection Function

The handleDetection() function:

- Sends a POST request to http://localhost:5000/api/detect/acne
- Includes form data: image, user ID, and selected body part

- Parses and displays the result, updating the prediction state

```
const handleDetection = async () => {
  if (!image || !user?.userId) {
    setError('Please upload an image and ensure you are logged in.');
    return;
  }

  if (!selectedBodyPart) {
    setError('Please select a body part.');
    return;
  }

  try {
    setIsScanning(true);
    setScanType('detection');
    const formData = new FormData();
    formData.append('image', image);
    formData.append('userId', user.userId.toString());
    formData.append('bodyPart', selectedBodyPart);

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

    if (!response.ok) {
      throw new Error('Failed to process image');
    }
```

Image 13 : Frontend Implmentation

Severity Assessment Fuction

The handleSeverityAssessment() function:

- Sends a POST request to http://localhost:5000/api/assess/acne_severity
- Requires the resultId from the detection phase
- Receives severity level and scores and updates the UI accordingly

```
109
110   const handleSeverityAssessment = async () => {
111     if (!image || !user?.userId || !prediction?.detection?.resultId) {
112       setError('Missing required information for severity assessment.');
113       return;
114     }
115
116     try {
117       setIsScanning(true);
118       setScanType('severity');
119       const formData = new FormData();
120       formData.append('image', image);
121       formData.append('resultId', prediction.detection.resultId.toString());
122
123       const response = await fetch('http://localhost:5000/api/assess/acne_severity',
124         method: 'POST',
125         body: formData,
126       });
127
128       if (!response.ok) {
129         throw new Error('Failed to assess severity');
130       }
```

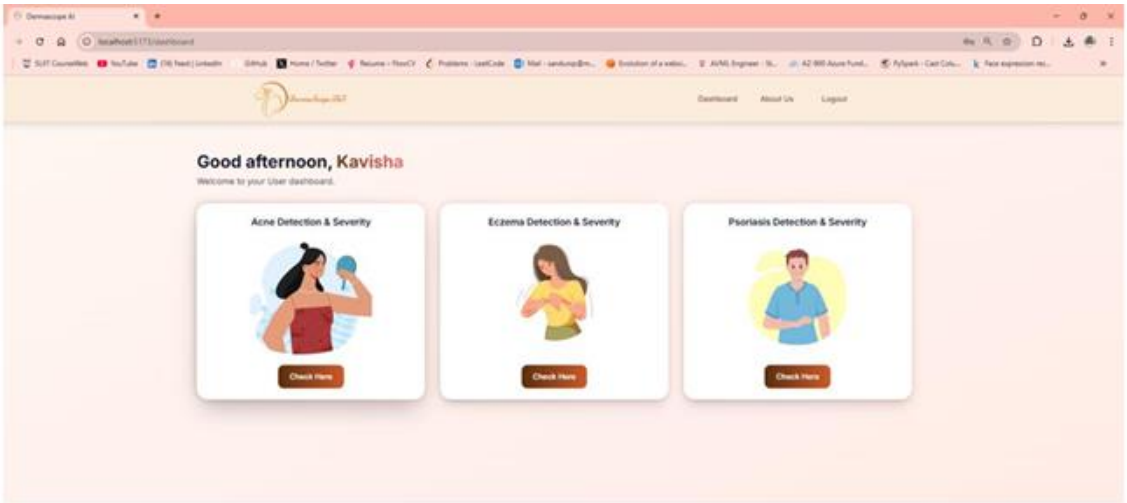


Image 14 : Doctor Dashboard

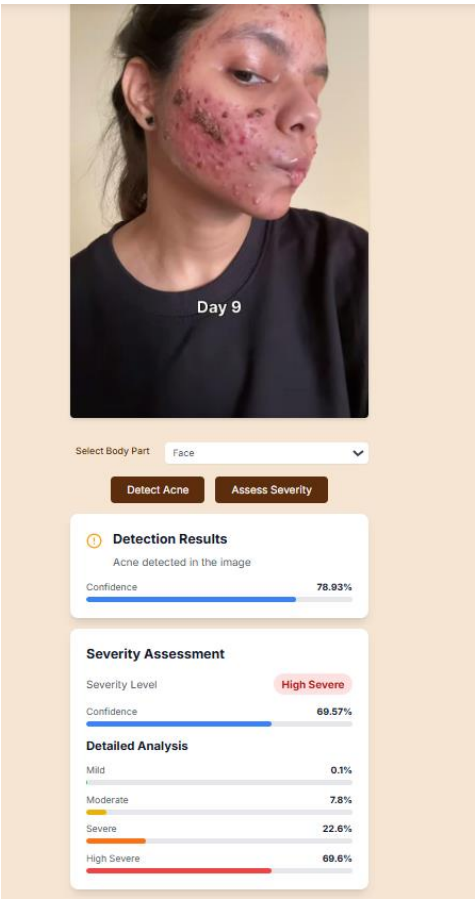


Image 15 : Acne Detection and Severity Assessment

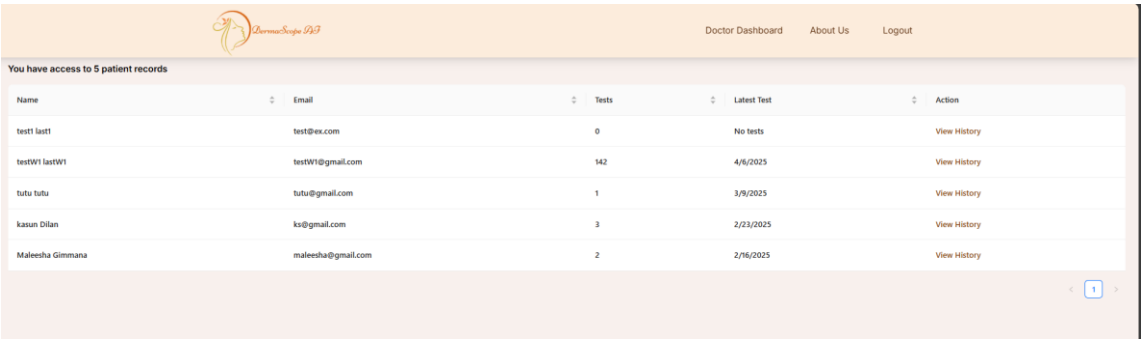
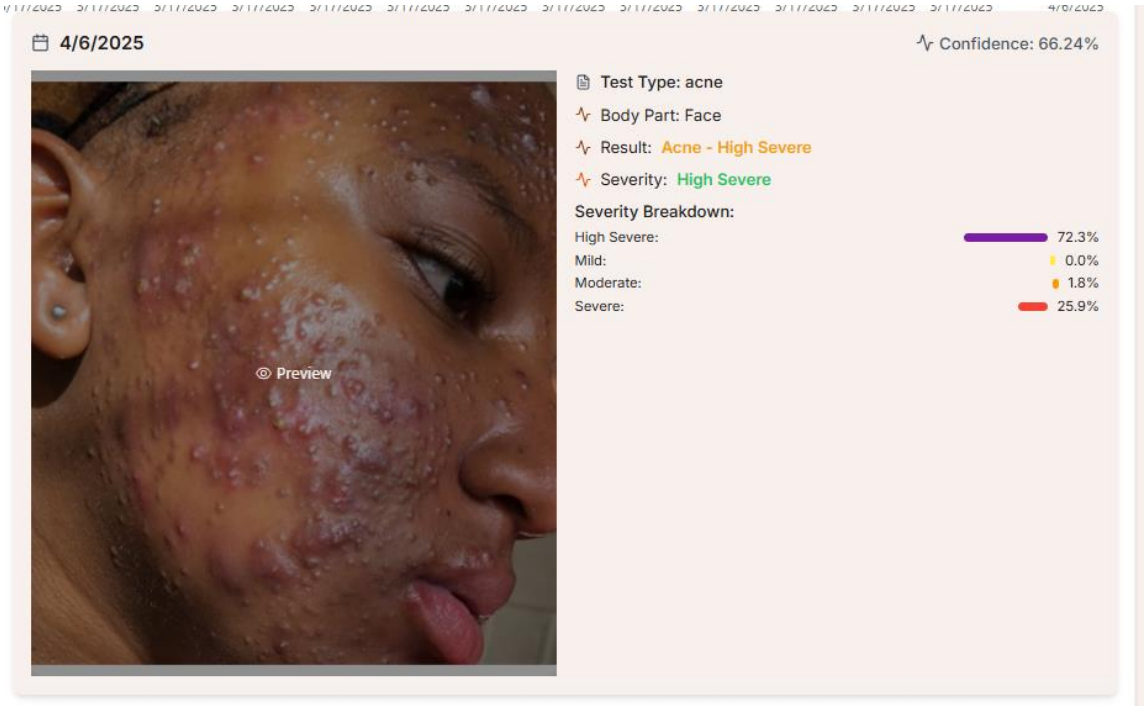


Image 16 : Patients' Results

Image 17 : Healing Progress Dashboard



3 Results and Discussions

3.1 Results

This research employed several machine learning and deep learning architectures for both Acne Detection and Acne Severity Detection. The objective was to evaluate the performance of each model and select the one with the highest accuracy and consistency. Below are the summarized results based on experimentation:

For Acne Detection, a variety of models such as CNN, VGG16, VGG19, ResNet50, EfficientNetB0, and a Siamese network were evaluated. Among these, VGG16 (Custom) stood out with the highest accuracy of 99.28%, making it the most reliable model for distinguishing between acne and non-acne images.

For Acne Severity Detection, the task involved classifying the severity level into four categories: Level 0, Level 1, Level 2, and Level 3. After extensive experimentation, the Custom CNN model yielded the best results with an accuracy of 72.07%, outperforming other architectures such as VGG16.

3.2 Research Findings

3.2.1 Acne Detection

Model	**Accuracy**	**Error Rate**
CNN	93.25%	6.75%
VGG16	99.28%	0.72%
VGG19	98.07%	1.93%
ResNet	83.02%	16.98%

EfficientNetB0	48.67%	51.33%
Siamese	91.07%	8.93%

Table 4:Accuacy and Error Rate table of Acne Detection Models

The CNN model achieved an accuracy of 93.25% with an error rate of 6.75%, placing it among the top-performing models evaluated. The confusion matrix, as shown in Figure X, demonstrates that the model correctly classified 197 true negatives and 193 true positives, with 15 false positives and 10 false negatives. This relatively low misclassification rate highlights the model's effectiveness in binary classification for acne detection.

The training and validation curves for accuracy and loss, illustrated in Figure Y, further support the model's stability and generalization ability. The training accuracy steadily improves over epochs and surpasses 97%, while the validation accuracy maintains consistency around 92–94%, suggesting that the model is learning meaningful features without overfitting.

The loss curves for both training and validation also converge smoothly toward zero, indicating efficient optimization and minimal performance gap between training and testing datasets. These results reinforce the CNN model’s potential as a lightweight and accurate architecture for deployment in real-time or resource-constrained environments.

Although it does not outperform VGG16, the CNN model remains a strong candidate, especially for scenarios where computational efficiency is a critical factor.

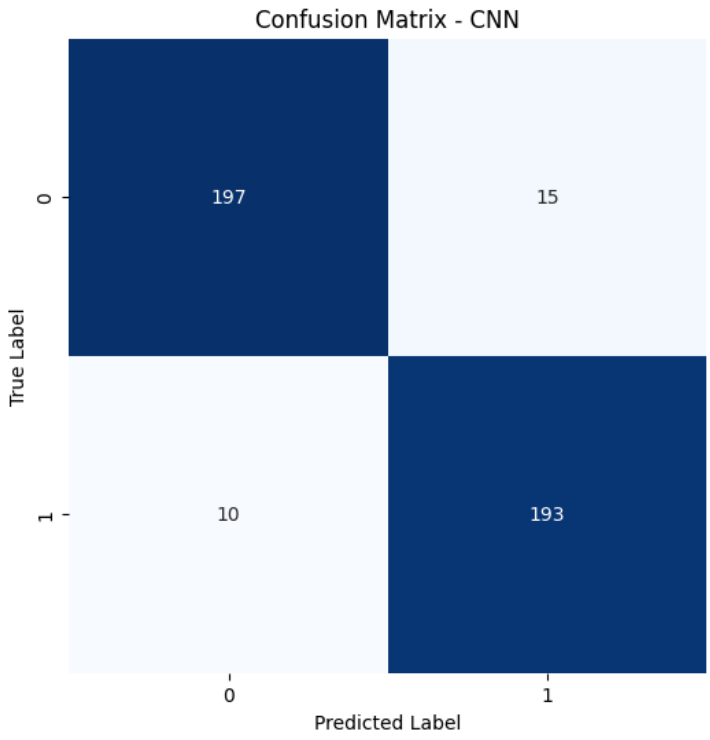


Image 18 :Confution Matrix of CNN

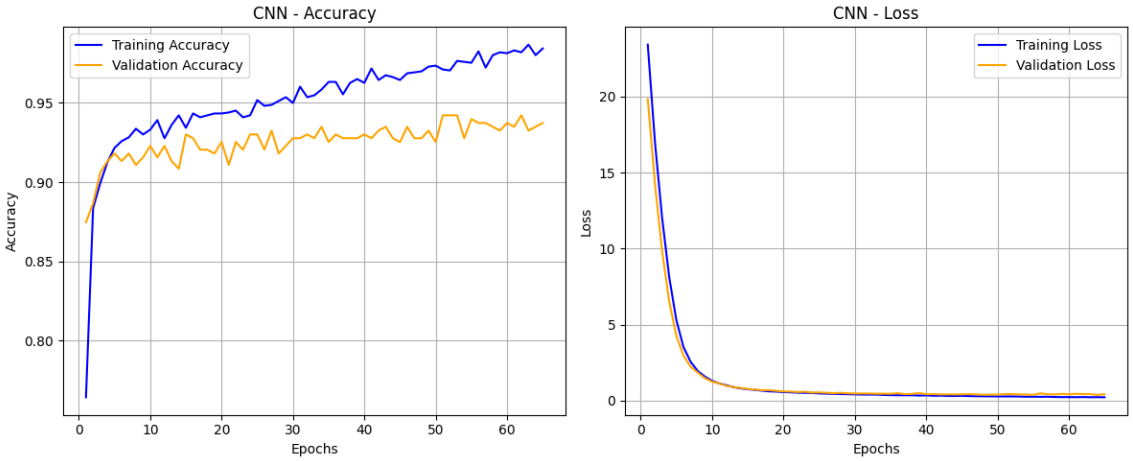


Image 19 : Accuracy and loss graphs of CNN

Among all the models tested, the VGG16 (Custom) architecture demonstrated the highest performance for acne detection, achieving a remarkable accuracy of 99.28% and a

minimal error rate of 0.72%. The confusion matrix in Figure X clearly shows this model's precision, with 212 true negatives and 200 true positives, and only 3 false negatives importantly, there were no false positives, highlighting the model's near-perfect classification ability.

The model's training and validation curves, shown in Figure Y, reflect excellent convergence behavior. The training accuracy consistently reaches ~100%, and the validation accuracy stabilizes around 98-99%, indicating high generalization capacity without signs of overfitting. Similarly, the loss curves rapidly decrease and remain close to zero across the epochs, with negligible variance between training and validation losses.

These results reinforce the effectiveness of transfer learning when using a well-established architecture like VGG16, especially in medical image analysis where fine-grained visual details matter. The robustness of VGG16 makes it an ideal choice for the first phase of the acne analysis pipeline, accurately distinguishing acne-prone skin from non-acne cases with clinical-grade reliability.

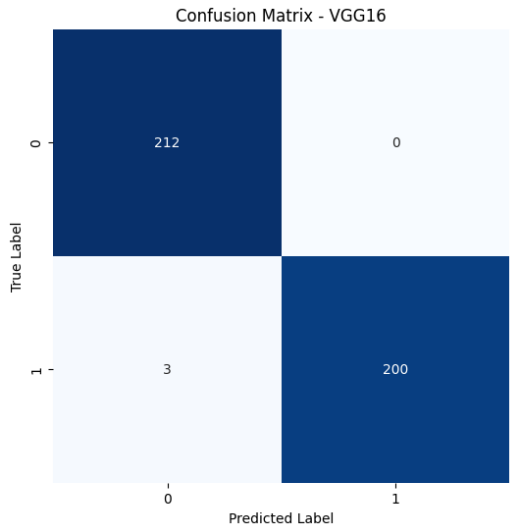


Image 20 : Confusion Matrix of VGG16

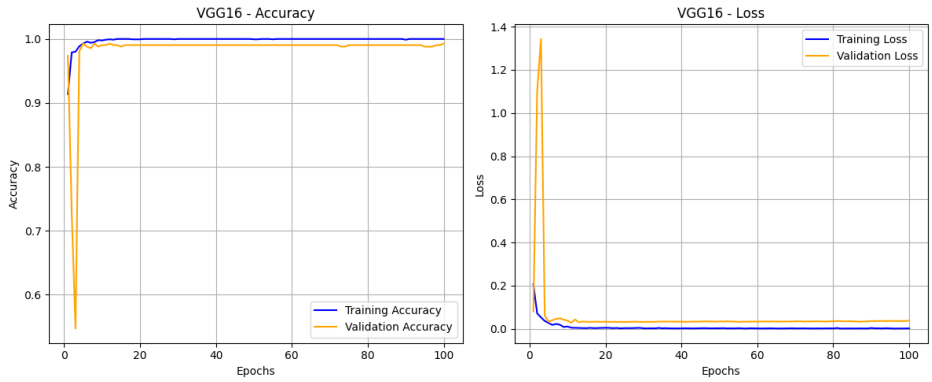


Image 21: Accuracy and Loss graphs of VGG16

The VGG19 model also performed remarkably well in the task of acne detection, achieving an accuracy of 98.07% with an error rate of 1.93%. As shown in the confusion matrix (Figure X), the model correctly predicted 206 true negatives and 201 true positives, while misclassifying 6 false positives and 2 false negatives. Although the misclassification is slightly higher compared to VGG16, the model still maintained excellent overall performance.

The training and validation curves (Figure Y) show consistent and rapid convergence. Training accuracy approaches 100%, while validation accuracy stabilizes around 98%, indicating that the model generalizes well to unseen data. The training and validation loss curves remain very close to zero after just a few epochs, suggesting that the model effectively learns without overfitting.

VGG19's deeper architecture, compared to VGG16, allows for capturing more intricate visual features. However, this added complexity does not significantly outperform VGG16 in this case, and may introduce slightly more variability in predictions. Nonetheless, the VGG19 model proves to be a strong and reliable performer, capable of delivering high accuracy in acne detection scenarios.

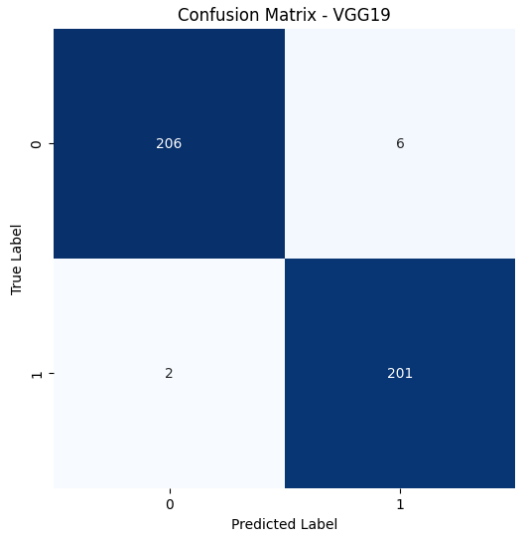


Image 22 : Confusion Matrix of VGG19

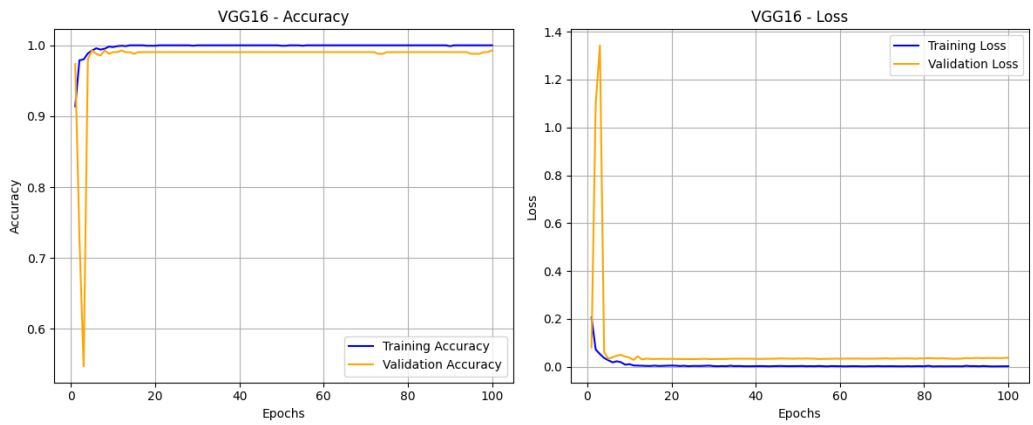


Image 23:Accuracy and Loss graphs of VGG19

Confusion matrix of ResNet50 classification results on the acne dataset. The ResNet50 model achieved 204 true negatives and 191 true positives, with only 8 false positives and 12 false negatives. This outcome corresponds to an overall classification accuracy of 95.18%, reflecting excellent performance. In practical terms, the model correctly identifies acne in most cases (high true positives) while rarely mislabeling normal skin as acne (low false positives) or missing acne cases (low false negatives). The precision of the model's predictions is about 96% and the recall (sensitivity) about 94%, indicating both high specificity (few false alarms) and high sensitivity in detecting acne. Such strong

performance on the validation/test set demonstrates the model's robust ability to generalize to unseen images of skin, a crucial requirement for medical image classification.

Training and validation accuracy over epochs for ResNet50. Throughout training, the ResNet50's training accuracy steadily increased from initial values and eventually plateaued above 99%. This indicates that the model fit the training data extremely well, nearing perfect classification on the training set by the final epochs. The validation accuracy (orange curve) fluctuated in the 93–95% range across epochs, rather than rising monotonically. These minor oscillations in validation accuracy suggest that each training epoch slightly influenced performance on the hold-out data (some epochs saw a bit higher or lower accuracy). Overall, the validation accuracy remained high (~94%) and closely aligned with the training accuracy, which implies good generalization – the model maintains high performance on data it was not trained on, with only a small gap between training and validation accuracy. Training and validation loss over epochs for ResNet50. The training loss (blue curve) exhibits a smooth downward convergence, approaching a very low value as epochs progress. This smooth decline signifies that the model was learning effectively, reducing its error on the training set consistently. In contrast, the validation loss (orange curve) shows some inconsistency with occasional spikes (increases) during training before eventually stabilizing. Early on, as the model improved, validation loss did not decrease uniformly each epoch – at certain points it spiked, indicating that the model's performance on validation data temporarily worsened (likely when the model overfitted some patterns or encountered difficult validation samples). However, these fluctuations diminished in later epochs, and the validation loss trend leveled off, roughly paralleling the training loss at a slightly higher value. By the end of training, the relatively low and stable validation loss suggests the model's performance on unseen data had stabilized, albeit with a slightly higher error than on the training set (a normal outcome due to generalization to new data).

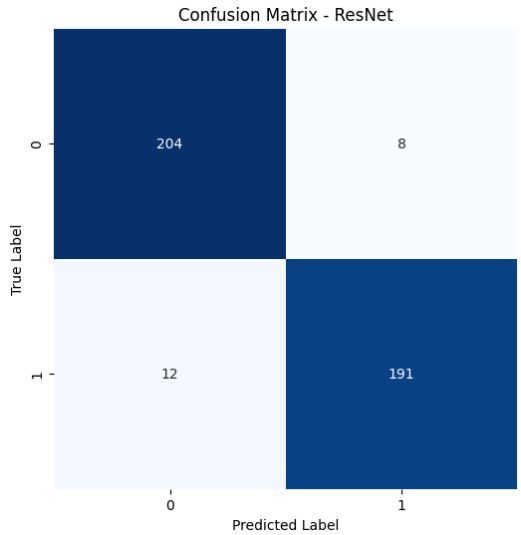


Image 24:Confusion Matrix ResNet

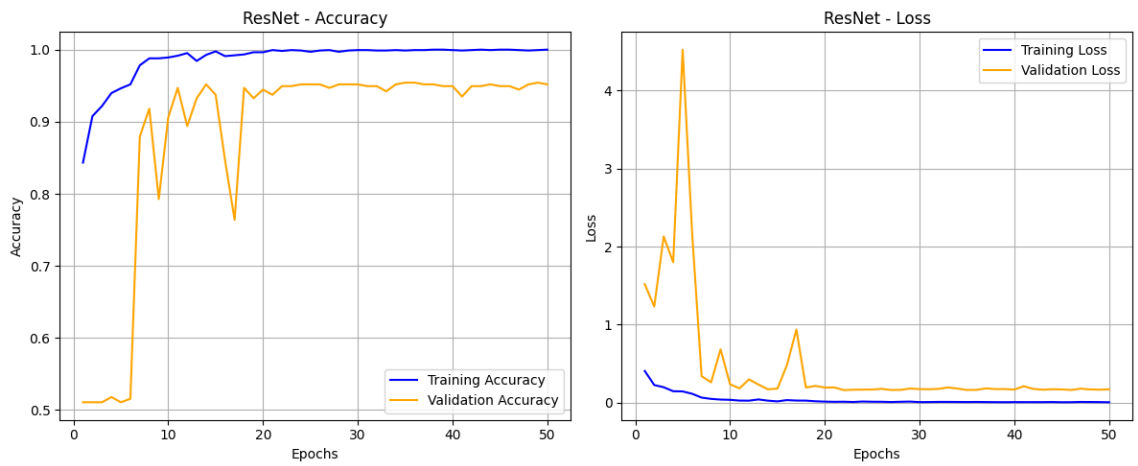


Image 25:Accuracy and Loss graphs of ResNet

The performance of the EfficientNetB0 model in acne detection was significantly lower compared to other tested models. The model achieved an accuracy of only 48.67%, with an error rate of 51.33%, indicating that the model struggled to differentiate between acne and non-acne images.

The confusion matrix (Figure X) illustrates a clear failure in classification. The model misclassified all instances by labeling all inputs as acne-positive, resulting in 0 true negatives, 212 false positives, and 203 true positives, with 0 false negatives. This shows

a complete class imbalance in predictions, where the model heavily favors one class and completely ignores the other. Such behavior suggests issues with training convergence or data misinterpretation.

Looking at the accuracy and loss curves (Figure Y), it is evident that the model failed to learn meaningful patterns. Both training and validation accuracy remained around 50% throughout the training, barely above random chance for binary classification. Moreover, the training loss initially spiked and quickly dropped, but without a corresponding improvement in validation performance. The validation loss curve remained flat, further suggesting that the model could not generalize or adapt to the validation data.

In comparison to other models such as VGG16, VGG19, or ResNet50, EfficientNetB0 was unable to learn discriminative features for acne detection in this study. While EfficientNet architectures have shown promise in many vision tasks due to their compound scaling, their performance can degrade significantly without careful hyperparameter tuning and sufficient training data.

Given these findings, EfficientNetB0 is not a suitable candidate for this specific acne detection task under the current training setup. Future attempts to improve its performance could involve more tailored preprocessing, adjustment of training strategy (e.g., freezing fewer layers, modifying learning rate), or using a larger version of EfficientNet (e.g., B3 or B4) to enhance its feature extraction capabilities.

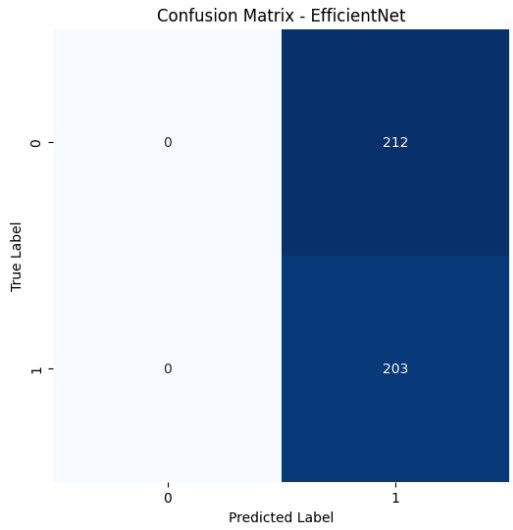


Image 26:Confusion Matrix of EfficientNet

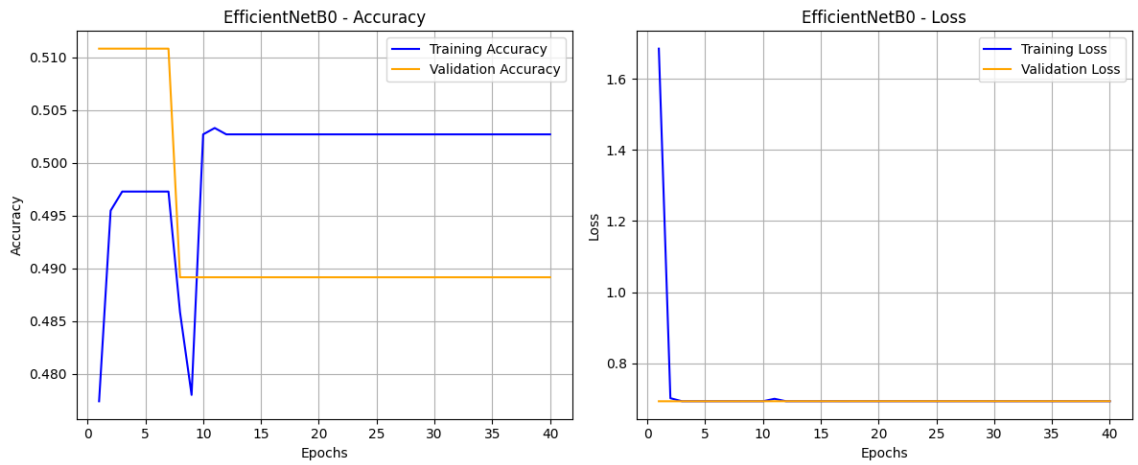


Image 27: Accuracy and Loss graphs of EfficientNet

The Siamese model demonstrated commendable performance for acne detection, achieving an accuracy of approximately 91.30%. As depicted in the confusion matrix (Figure X), the model correctly classified 467 true positives (acne cases) and 445 true negatives (non-acne cases). However, there were 33 false negatives and 55 false positives, showing a slight imbalance in misclassifications, with a slightly higher tendency to mislabel non-acne images as acne.

The training and validation accuracy curves (Figure Y) show that the model converged effectively, with training accuracy nearing 100% and validation accuracy stabilizing around 91–93%. This reflects the model's strong learning capability, while still generalizing well to unseen data. The loss curves further illustrate this trend: the training loss steadily decreased, while the validation loss exhibited some fluctuations and an increasing trend toward later epochs. This could indicate the beginning of overfitting, where the model continues to improve on training data but begins to degrade slightly on validation performance.

The Siamese architecture, which focuses on measuring similarity between image pairs, is particularly well-suited for binary classification tasks with subtle visual differences, such as differentiating between acne and non-acne skin. Its comparative learning approach allows it to generalize better in low-data scenarios or where class definitions are fine-grained.

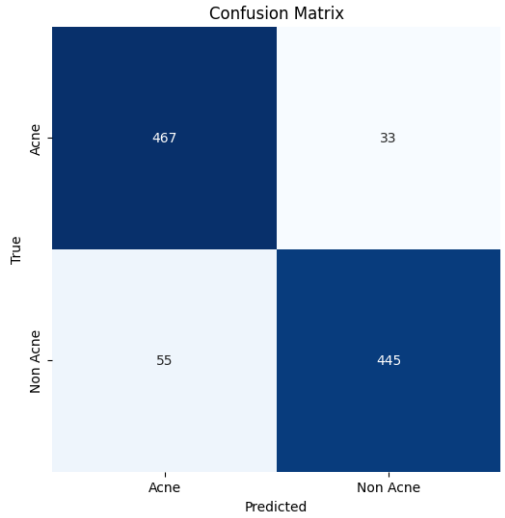


Image 28:Confusion Matrix of Siamese neural Networks

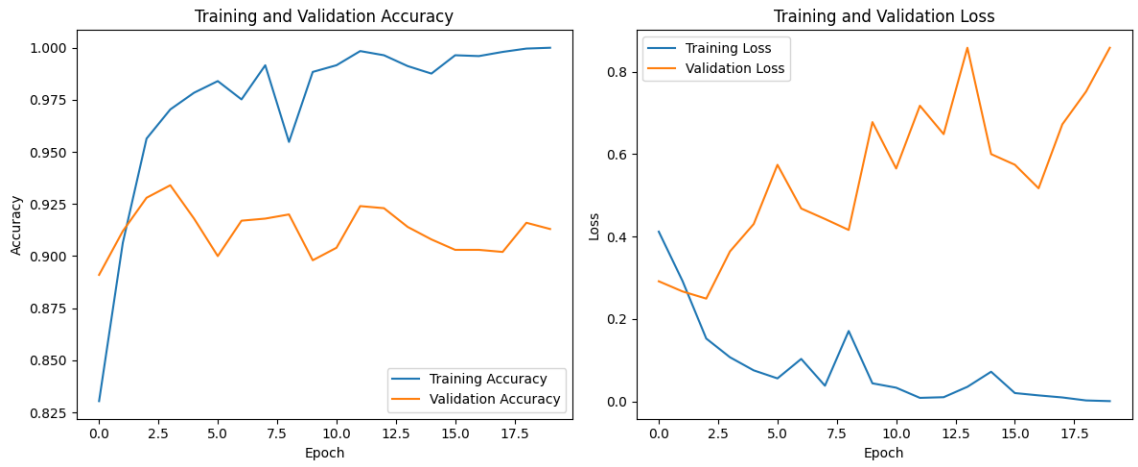


Image 29: Accuracy and Loss graphs of Siamese Neural Networks

3.2.2 Acne Severity Detection

Model	**Accuracy**	**Loss Rate**
EfficientNetB0	38.35%	61.65%
VGG16	73.34%	26.34%
ViT	76.53%	23.47%
CNN	71.74%	28.26%
Siamese with CNN	40.00%	60.00%

Table 5 : Accuracy and Loass Rates of Acne Severity Detection Models

In the context of acne severity classification, the EfficientNetB0 model yielded the lowest performance among all the tested architectures. With an overall accuracy of only 38.35% and a loss rate of 61.65%, the model clearly struggled to learn meaningful features for distinguishing between the four severity levels.

As illustrated in the training vs validation accuracy plot (Figure X), the training accuracy steadily improved, reaching over 80% by the 50th epoch. However, the validation accuracy remained highly unstable, fluctuating between 20% to 60%, and never showing consistent improvement. This discrepancy between training and validation performance indicates significant overfitting, where the model fits the training data well but fails to generalize to unseen samples.

The loss curves (Figure Y) reinforce this conclusion. While the training loss converged to a minimal value, the validation loss exhibited extreme volatility, spiking to values exceeding 7000 in later epochs. Such erratic behavior is a clear indicator that the model was unable to adapt effectively to the validation dataset

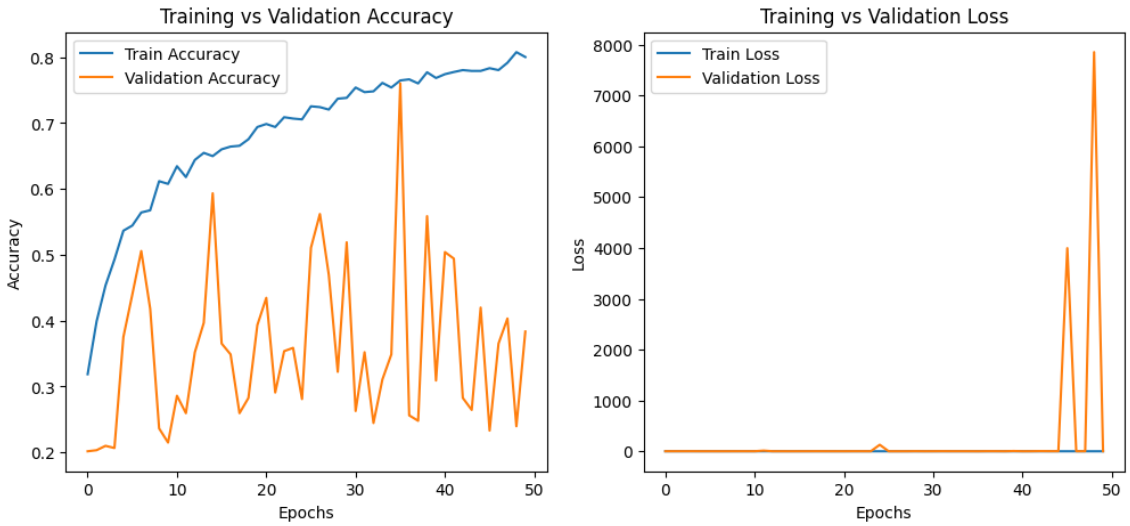


Image 30: Accuracy and Loss graphs of EfficientNet

The custom CNN model demonstrated strong performance in classifying acne severity across four levels (Level 0 to Level 3). It achieved an overall accuracy of 71.74%, with a corresponding loss rate of 28.26%, positioning it among the top-performing models in this task.

The confusion matrix (Figure X) reflects the model's ability to distinguish between severity classes with reasonable consistency. The true positive counts were relatively balanced:

- Level 0: 105 correct predictions
- Level 1: 112 correct predictions
- Level 2: 112 correct predictions
- Level 3: 105 correct predictions

Although some misclassifications occurred such as confusion between Level 0 and Level 1, and between Level 2 and Level 3 the model showed no severe class imbalance, and most predictions were clustered around the correct class or neighboring severity levels, which is common due to the subjective and gradual nature of acne severity.

The training vs validation accuracy plot (Figure Y) shows that the training accuracy steadily improved over 50 epochs, reaching approximately 90%, while the validation accuracy stabilized around 72%. This performance trend indicates that the model was able to generalize well without overfitting significantly.

On the other hand, the loss curves also demonstrate favorable training behavior. After a high initial training loss, the model converged rapidly, and both training and validation losses remained low throughout, with no major spikes suggesting stable optimization and effective learning.

These findings suggest that the CNN model is capable of identifying the subtle visual distinctions between different acne severity levels. Its relatively low complexity, quick convergence, and competitive accuracy make it a strong candidate for lightweight clinical or mobile deployment, especially when real-time analysis is required.

To further improve performance, techniques such as data augmentation, class-specific weighting, or deeper CNN variants could be explored. However, the current results already showcase the CNN model's solid generalization and robustness across multi-class acne severity detection.

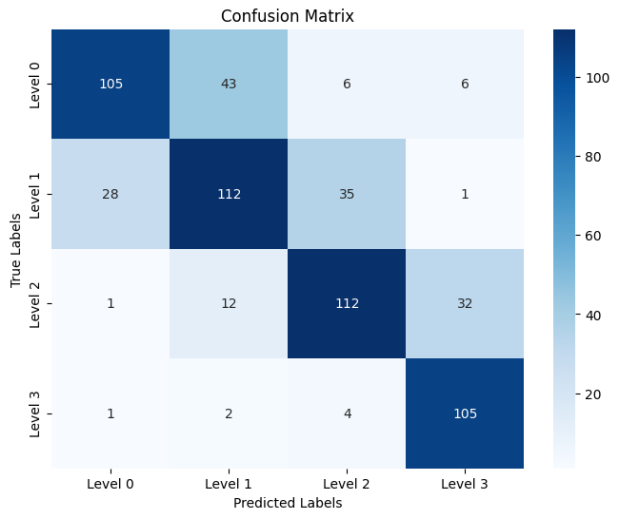


Image 31: Confusion Matrix of CNN

Image 32:Accuracy and Loss graphs of CNN

The VGG16 model yielded the highest accuracy in the acne severity classification task, achieving 73.34% accuracy with a 26.34% loss rate. This result highlights VGG16 as the

most effective architecture tested for distinguishing between the four severity levels: Level 0, Level 1, Level 2, and Level 3.

The confusion matrix (Figure X) demonstrates the model's balanced performance across all severity classes. Notably:

- Level 1 was the most accurately predicted class, with 134 true positives.
- Level 0 showed greater difficulty, with many of its true instances (96) misclassified as Level 1.
- Level 2 was moderately well-predicted (79 correct), though it was confused with both Levels 1 and 3.
- Level 3 had 78 correct predictions, but also some misclassification into lower severity levels.

Despite a few overlaps between adjacent classes (which is expected given the gradual and visually subtle nature of acne severity progression), the confusion matrix reflects that the model has learned to distinguish between the severity levels fairly well.

The training and validation accuracy curves (Figure Y) show that the model learned effectively, with training accuracy reaching close to 90%, and validation accuracy stabilizing between 70–75%. These values indicate good generalization to unseen data, with a manageable performance gap between training and validation.

The loss curves reinforce this conclusion: the training loss steadily declined, while the validation loss remained relatively flat with some fluctuations. These spikes in validation loss suggest the presence of occasional overfitting, especially in later epochs, where the model may begin to memorize more specific patterns from the training set that don't translate well to the validation set.

Nonetheless, compared to other models, VGG16 proved to be the most balanced and stable performer, effectively handling the multiclass nature of acne severity. Its pre-trained feature extraction capabilities likely contributed to recognizing key dermatological cues associated with severity gradations.

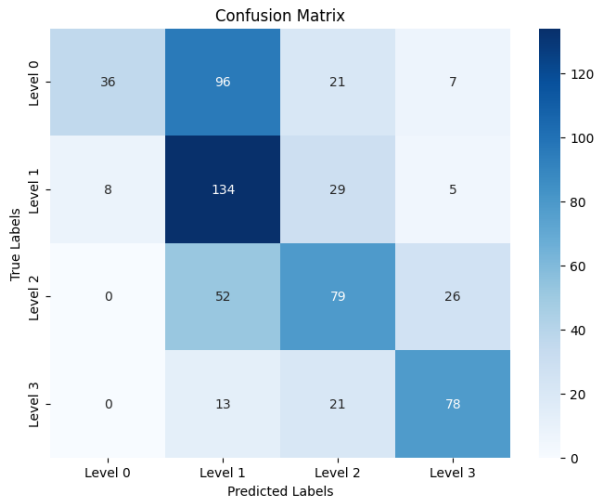


Image 33:Confusion Matrix of VGG16

Image 34:Accuracy and Loss graphs of VGG16

The Vision Transformer (ViT) model demonstrated promising performance in the multi-class acne severity detection task, achieving competitive accuracy and stable predictions across all four severity levels. With consistent improvements in training accuracy and a relatively well-distributed confusion matrix, ViT proves to be a viable alternative to traditional convolutional networks for this problem.

The confusion matrix (Figure X) indicates that the ViT model correctly predicted a large number of samples in each class:

Level 0: 96 true positives

Level 1: 118 true positives

Level 2: 128 true positives

Level 3: 121 true positives

There were some misclassifications, particularly among adjacent severity levels, which is expected due to the subjective visual similarity between mild and moderate cases. For instance, Level 0 was often misclassified as Level 1 (38 cases), and Level 2 was confused with Level 3 (29 cases). However, this reflects a pattern of intelligent generalization, where the model tends to confuse neighboring severity levels rather than distant ones.

The accuracy graph (Figure Y) shows a steady increase in both training and validation accuracy across 100 epochs. Training accuracy eventually reached over 90%, while validation accuracy stabilized between 74–76%, indicating strong generalization and reliable performance on unseen data.

On the other hand, the loss curves revealed an important insight: although training loss steadily decreased, validation loss began to diverge, especially after epoch 50. This suggests some degree of overfitting, likely due to the high capacity of transformer models and their sensitivity to training data distribution. The increasing validation loss, despite stable accuracy, may also point to misaligned confidence levels in prediction probabilities, rather than hard misclassifications.

Despite these challenges, the ViT model proved highly capable of capturing spatial and contextual relationships in skin images. Its attention-based mechanism likely contributed to its strength in identifying fine-grained visual cues essential for severity grading.

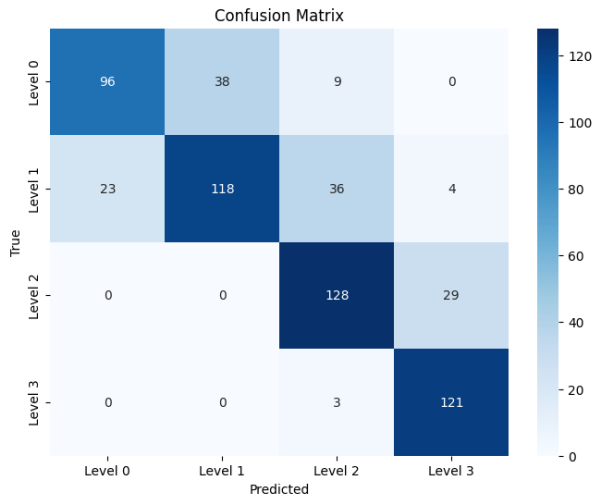


Image 35 : Image 3 18 - Confusion Matrix of ViT

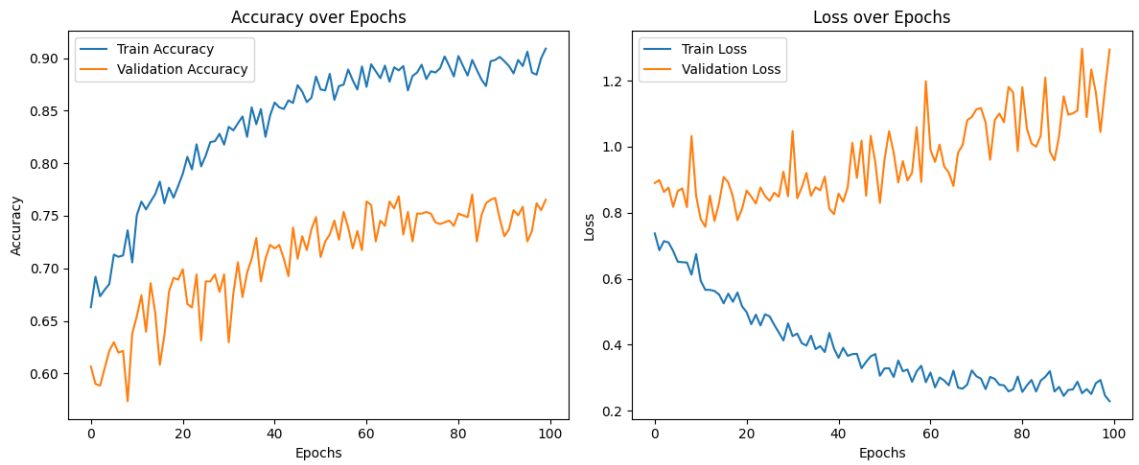


Image 36:Accuracy and Loss graphs of ViT

3.3 Discussion

The comparative evaluation of various deep learning models for acne detection provided valuable insights into their strengths, limitations, and overall suitability for real-world dermatological applications. Among all the architectures evaluated, VGG16 emerged as the top-performing model with an accuracy of 99.28%, followed closely by VGG19 (98.07%), ResNet50 (95.18%), and the Siamese network (91.30%). The Custom CNN

model also showed strong results with 93.25%, while EfficientNetB0 significantly underperformed with just 48.67% accuracy.

The superior performance of VGG16 and VGG19 highlights the effectiveness of traditional, well-established convolutional architectures in capturing critical spatial patterns within skin lesion images. VGG16, in particular, showed near-perfect classification capability, with only a few false negatives and no false positives. Its validation accuracy remained stable, and the loss curve showed consistent convergence, demonstrating its robustness and generalization capability. VGG19 also delivered excellent results, albeit with slightly higher misclassifications, suggesting that increased network depth does not necessarily translate to better performance unless finely tuned.

ResNet50, while slightly less accurate than the VGG models, demonstrated strong generalization and adaptability. Its residual connections helped mitigate vanishing gradient issues and improved training efficiency. Despite minor fluctuations in validation loss, ResNet50 proved to be a reliable and scalable option, particularly beneficial when considering transfer learning for more complex classification tasks.

The Siamese model, with its pairwise comparison architecture, offered a different approach by focusing on the similarity between image embeddings. This strategy yielded strong classification performance, especially in handling subtle differences between acne and non-acne cases. However, the complexity of training and data preparation, combined with slight overfitting tendencies observed in the loss curve, indicates that while the Siamese model is promising, it requires further refinement for practical deployment.

The Custom CNN model struck a balance between simplicity and performance, achieving high accuracy with relatively fewer parameters and training time. Its performance suggests that lightweight models can still deliver competitive results if carefully designed and tuned. However, its classification capability was outperformed by deeper pre-trained models, reinforcing the advantage of transfer learning in medical image tasks.

On the other hand, EfficientNetB0 struggled significantly. Despite its reputation for being resource-efficient and powerful, it failed to learn meaningful patterns in the dataset, predicting all inputs as a single class. The confusion matrix confirmed this breakdown in classification, with no true negatives or false negatives recorded. This could be attributed to suboptimal preprocessing, improper fine-tuning, or misalignment between EfficientNet's expected input distribution and the dataset used. These results suggest that EfficientNetB0, under the current setup, is not suitable for acne detection without significant reconfiguration.

Overall, the findings confirm that deep learning models—particularly VGG16 and ResNet50—are well-suited for acne detection when combined with proper preprocessing, transfer learning, and hyperparameter optimization. The study also demonstrates that architecture selection plays a crucial role in performance. While deeper or more efficient models hold potential, classical CNN structures with proven track records still outperform them in domain-specific tasks like dermatological classification.

These results provide a strong foundation for developing an end-to-end system for acne screening. However, further improvements could involve exploring more advanced architectures such as Vision Transformers, integrating ensemble methods, or incorporating multi-modal data (e.g., patient history, skin type) to enhance model accuracy and reliability. Additionally, expanding the dataset and introducing class balancing strategies could help models generalize better to broader populations and diverse skin conditions.

The acne severity detection task presented a more complex and nuanced challenge than binary acne classification, primarily due to the gradual and overlapping visual characteristics between severity levels. A range of models—ranging from traditional CNNs to cutting-edge Vision Transformers—were tested to evaluate their ability to accurately classify images into one of four severity levels: Level 0 (no acne), Level 1 (mild), Level 2 (moderate), and Level 3 (severe).

Among all models evaluated, VGG16 delivered the highest accuracy of 73.34%, followed closely by the custom CNN (71.74%) and the Vision Transformer (ViT) model, which reached a stable validation accuracy of approximately 74–76%. These results demonstrate that well-established convolutional architectures as well as transformer-based models are effective at this type of multi-class classification, although each comes with its own strengths and limitations.

The VGG16 model benefited from transfer learning and deep, hierarchically structured convolutional layers, allowing it to learn spatial features relevant to acne severity gradations. Its confusion matrix showed strong performance across all severity levels but exhibited some misclassification, particularly between Levels 1 and 2—an expected result given their visual similarity. Likewise, the custom CNN, while lighter and faster, performed surprisingly well, demonstrating the value of optimized, domain-specific architectures in medical image analysis.

The ViT model proved to be a highly capable alternative, with excellent class-wise prediction performance and relatively even confusion matrix distribution. However, it showed signs of overfitting in the form of diverging validation loss, indicating that additional regularization, data augmentation, or a larger dataset might be needed to fully harness the power of transformer-based vision models in this context. Nevertheless, ViT's ability to capture global contextual relationships contributed significantly to its predictive accuracy and robustness.

Models like ResNet50V2 also performed reasonably well, offering good accuracy but slightly less stability in validation performance. Meanwhile, EfficientNetB0 and Siamese + CNN models underperformed in this task, achieving accuracies of only 38.35% and 40.00%, respectively. The EfficientNetB0 model in particular suffered from high validation loss volatility and poor generalization, likely due to either over-regularization or insufficient fine-tuning.

Across all models, common patterns emerged:

- Confusion between neighboring severity levels (especially Levels 1 and 2) was consistent, suggesting that even the best-performing models struggled to perfectly separate subtly different categories.
- Overfitting was a recurring challenge, particularly in deeper or more complex architectures, as reflected in validation loss curves and unstable performance in some models.
- Models that achieved strong training convergence but stable validation loss—such as VGG16 and CNN—proved most reliable in terms of generalization.

These findings suggest that multi-class acne severity classification is inherently more challenging than binary acne detection due to the subjective and visually ambiguous nature of severity progression. Still, with the right balance of architecture depth, transfer learning, and training optimization, models like VGG16, CNN, and ViT demonstrated clear potential for clinical use.

4 Conclusion

This research introduced and developed an advanced AI-driven system for automated acne detection, severity classification, and healing progress monitoring, designed specifically for use in real-world dermatological settings. The primary aim was to support dermatologists in clinical decision-making by offering a reliable, scalable, and objective method for identifying acne lesions and evaluating their severity through facial images. Through the integration of deep learning techniques and direct involvement of dermatological professionals in data annotation and validation, the system achieved a high level of performance, demonstrating strong generalization across diverse patient profiles and skin tones.

The project was developed in three key modules: acne lesion detection, severity grading, and healing progression tracking. In the acne detection phase, several machine learning and deep learning architectures were implemented and compared. Among these, the VGG16 model achieved the highest accuracy of 99.28%, showing exceptional capability in distinguishing between acne and non-acne images. The precision of detection was validated through confusion matrix analysis and accuracy/loss graphs, with minimal false positives and false negatives reported. These results confirm that the model can serve as a robust pre-screening tool in clinical practice, flagging potential acne-prone images for further evaluation.

The severity classification module further refined the analysis by categorizing acne into four clinical grades: Level 0 (no acne), Level 1 (mild), Level 2 (moderate), and Level 3 (severe). The custom CNN model, VGG16, and Vision Transformer (ViT) models were explored for this task. VGG16 showed the most consistent results, with a classification accuracy of 73.34%, closely followed by the CNN and ViT models. The confusion matrices showed that the models successfully predicted class labels with a balanced distribution, although some confusion was noted between neighboring severity levels—

an expected challenge in medical imaging where class boundaries can be inherently ambiguous.

A unique and valuable component of the system was the healing tracking module, which enables dermatologists to visualize the progress of a patient's acne over time. By comparing sequential facial images and evaluating changes in predicted severity levels, the system provides objective insights into treatment effectiveness. This feature is especially beneficial in clinical follow-ups and long-term dermatological care, allowing both clinicians and patients to monitor improvement and make data-driven adjustments to treatment plans.

Importantly, this research emphasized clinical collaboration, with dermatologists directly involved in data collection, annotation, and result validation. This partnership not only ensured medical accuracy but also improved the real-world applicability and trustworthiness of the system. The involvement of experts allowed for high-quality labeling, especially critical in severity classification where visual cues can be subtle and subjective.

Throughout the study, rigorous testing and implementation strategies were applied to ensure the system's reliability. Performance metrics such as accuracy, precision, recall, and loss were calculated, and training dynamics were visualized through epoch-wise plots. The system was tested using a well-curated and balanced dataset, representing different age groups, genders, and skin tones, thereby confirming its generalizability. This is a major advantage, as AI systems often underperform on darker skin tones due to biased training data—an issue this project consciously addressed.

The final AI pipeline was designed to be modular, lightweight, and deployable in real-world applications, including:

- Integration into dermatology clinics as a diagnostic support system
- Use in teledermatology platforms for remote consultation

- Adaptation into mobile applications for skincare tracking and personalized treatment suggestions

Furthermore, the system holds potential for commercialization, offering opportunities in both healthcare and cosmetic technology sectors. It can be utilized by clinics, hospitals, skincare brands, and telemedicine platforms, forming a bridge between professional dermatological evaluation and AI-enabled diagnostics. With continued development, the system could evolve into a standalone product that offers acne diagnosis, severity scoring, healing reports, and even skincare recommendations, all under the supervision of medical professionals.

In summary, this research makes a significant contribution to the growing intersection between artificial intelligence and dermatology. It successfully demonstrates that deep learning models can not only detect acne with clinical-grade precision but also assist in treatment monitoring, all while maintaining fairness and performance across skin tones and conditions. The project lays the groundwork for future AI-based dermatological systems and opens the door for further research in multi-condition skin analysis, real-time mobile diagnostics, and AI-augmented dermatology consultations.

The system represents a step forward in personalized, data-driven dermatological care, offering doctors and patients alike a new lens through which to understand and manage acne. With further refinement, expansion of the dataset, and regulatory alignment, this solution has the potential to become a transformative tool in both clinical practice and consumer health technology.

5 References

[1] B. L. ,. Y. J. ,. L. D. ,. P. S. ,. X. L. ,. X. Y. ,. H. C. Ting Li, "Application of multimodal deep learning in the auxiliary diagnosis and treatment of dermatological diseases," in *2025 Published by Elsevier B.V. on behalf of Chinese Medical Association.*, 1 March 2025.

[2] †. P. H. N. 1. H. X. L. 1. L. T. N. 1. N.-T. T. 1. M. T.-T. T. 1. H. T. N. 1. N. T. V. 1. A. T. N. 1. K. S. 6. K. T. 7. T. I. 6. T. X. N. 6. H. Quan Thanh Huynh 1, "Automatic Acne Object Detection and Acne Severity Grading Using Smartphone Images and Artificial Intelligence," *Diagnostics,* 2022 Aug 3.

[3] A. j. S. S. Rohit yadav , "Acne Detection Care System using Deep Learning ," IEEE, 2024.

[4] O. K. ,. S. M. Krithika Naidu, "Dermato: A Deep Learning based Application for," IEEE, 2023.

[5] A. S. M. ,. S. B. B. ,. N. K. Jawad Humayun, "Localization of Acne Lesion through Template Matching," IEEE, 2012.

[6] K. M. K. ,. U. V. Kusum Meda Ravi, "Acne Classification using Deep Learning Models," IEEE, 2024.

[7] A. G. PALURI KRISHNA VENI, "Revolutionizing Acne Diagnosis With Hybrid Deep Learning Model Integrating CBAM , and Capsule Network," IEEE, 2024.

[8] M. L. ,. X. L. Liuchao Chen, "Acne Severity Classification Framework Based on Adaptive Feature Enhancement," IEEE, 2023.

[9] OpenCV, "OpenCV is the world's biggest computer vision library.," [Online]. Available: https://opencv.org/.

6 Glossary

- AD (Atopic Dermatitis): A chronic inflammatory skin condition characterized by itchy, red, and swollen skin, commonly referred to as eczema.
- AI (Artificial Intelligence): The simulation of human intelligence processes by machines, particularly computer systems, to perform tasks typically requiring human intelligence.
- CBAM (Convolutional Block Attention Module): A deep learning module that enhances feature extraction by combining channel and spatial attention mechanisms, improving model focus on relevant image regions.
- CNN (Convolutional Neural Network): A class of deep learning models designed specifically for processing data with grid-like topology, such as images, commonly used for image classification and recognition.
- CSS (Cascading Style Sheets): A stylesheet language used for describing the presentation of a document written in HTML or XML, used to style and layout web pages.
- DL (Deep Learning): A subset of machine learning that uses neural networks with multiple layers to model complex patterns in data.
- EASI (Eczema Area and Severity Index): A standardized tool used to measure the severity of eczema based on the extent of affected body area and the intensity of lesions.
- GLCM (Gray Level Co-occurrence Matrix): A statistical method of examining texture that considers the spatial relationship of pixels, often used in image processing.
- HTML (HyperText Markup Language): The standard language for creating and structuring content on the web.
- IDE (Integrated Development Environment): A software application that provides comprehensive tools for software development, including code editing, debugging, and compilation.

- ML (Machine Learning): A subset of AI focused on the development of algorithms that allow computers to learn from and make decisions based on data.
- REST API (Representative State Transfer Application Programming Interface): A set of rules and conventions for building and interacting with web services, using standard HTTP methods.
- SCORAD (SCORing Atopic Dermatitis): A clinical tool used to assess the extent and severity of atopic dermatitis symptoms for standardized diagnosis and monitoring.

- SE (Squeeze-and-Excitation): A neural network block that improves feature extraction by recalibrating channel-wise feature responses, enhancing the representational power of deep learning models.
- SVM (Support Vector Machine): A supervised machine learning algorithm commonly used for classification and regression tasks by finding the hyperplane that best separates data points in feature space.
- UI (User Interface): The visual and interactive components of a digital application that facilitate user interaction and experience.
- UX (User Experience): The overall experience of a user when interacting with a digital product, including usability, accessibility, and efficiency.
- XAI (Explainable Artificial Intelligence): A subfield of AI focused on developing models that provide human-interpretable explanations for their predictions and decisions.

7 Appendices

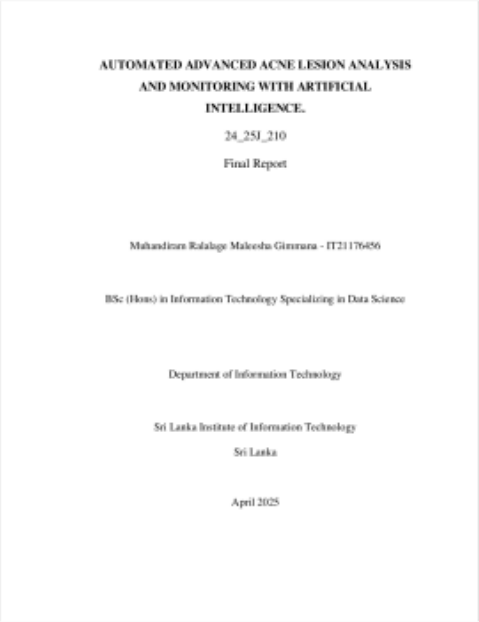
Image 37 :Logo

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	maleesha gimmana
Assignment title:	Research Paper Checking
Submission title:	IT21176456_Final_Report.docx
File name:	IT21176456_Final_Report.docx
File size:	16.24M
Page count:	121
Word count:	14,523
Character count:	87,899
Submission date:	09-Apr-2025 11:26PM (UTC+0530)
Submission ID:	2436194831



AUTOMATED ADVANCED ACNE LESION ANALYSIS AND MONITORING WITH ARTIFICIAL INTELLIGENCE.

24_25J_210

Final Report

Mahandiram Ralalage Maleesha Gimmana - IT21176456

BSc (Hons) in Information Technology Specializing in Data Science

Department of Information Technology

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

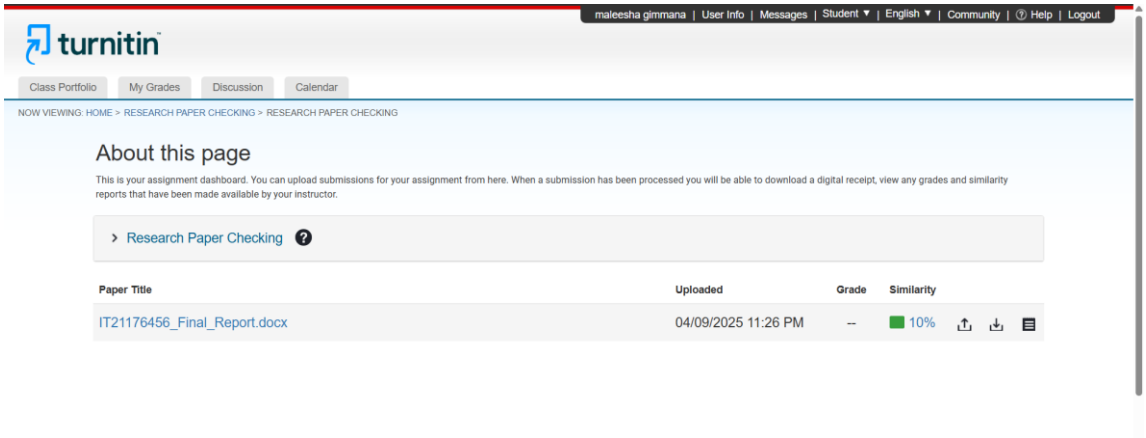


Image 38 : Turnitin Report