NITTE EDUCATION TRUST | N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India

"MITIGATING BIASES IN DERMATOLOGICAL DIAGNOSIS THROUGH GAN AUGMENTATION AND CLASSIFICATION"

Project Report submitted by

Kishor S Naik (4NM21IS068)
Neil Mascarenhas (4NM21IS095)
Prajwal P (4NM21IS105)
Nihal (4NM22IS405)

Under the Guidance of

Ms. Ashwitha C Thomas
Assistant Professor Gd-I
Department of Information Science & Engineering

In partial fulfillment of the requirements for the award of
Bachelor of Engineering in Information Science and Engineering

Department of Information Science Engineering
NMAM Institute of Technology, Nitte - 574110
(An Autonomous Institution affiliated to VTU, Belagavi)

November 2024

NITTE EDUCATION TRUST | N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work entitled

"Mitigating Biases in Dermatological Diagnosis through GAN Augmentation and Classification "

is a bonafide work carried out by

Kishor S Naik (4NM21IS068)

Neil Mascarenhas (4NM21IS095)

Prajwal P (4NM21IS105)

Nihal (4NM22IS405)

in partial fulfillment of the requirements for the award of

Bachelor of Engineering Degree in Information Science and Engineering *prescribed by* ***Visvesvaraya Technological University, Belagavi*** *during the year* ***2024-2025****.*

It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the Bachelor of Engineering Degree.

Signature of the Guide	***Signature of the HOD***	***Signature of the Principal***

Semester End Viva Voce Examination

Name of the Examiners	**Signature with Date**
1.	
2.	

ACKNOWLEDGEMET

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

First and foremost, we would like to thank **Dr. NIRANJAN N CHIPLUNKAR**, Principal, NMAMIT, Nitte, for his moral support towards completing our project work.

We would like to thank **Dr. ASHWINI B**, Head of the Department, Information Science & Engineering, NMAMIT, Nitte, for her valuable suggestions and expert advice.

We also extend our cordial thanks to Project Mentors, **Dr. MANJULA GURURAJ RAO, Mr. VASUDEVA PAI, Dr. NAGANNA CHETTY, Dr. BALASUBRAMANI R, Mr. ABHISHEK RAO** and **Ms. TANZILA NARGIS** for their support and guidance.

We deeply express our sincere gratitude to our guide **Ms. ASHWITHA C THOMAS,** Assistant Professor Gd-I**,** Department of ISE, NMAMIT, Nitte, for her able guidance, regular source of encouragement and assistance throughout this Project work.

We thank our Parents, and all the faculty members of Department of Information Science & Engineering for their constant support and encouragement.

Last, but not the least, we would like to thank our peers and friends who provided us with valuable suggestions to improve our Project.

ABSTRACT

Skin conditions affect individuals of all skin tones, yet biases in dermatological diagnosis disproportionately impact those with darker skin, leading to significant healthcare disparities. This project proposes a comprehensive solution that integrates advanced data preprocessing, Generative Adversarial Network (GAN) augmentation, and Convolutional Neural Network (CNN) models to mitigate these biases. The initial phase involves meticulous preprocessing of diverse dermatological image datasets, ensuring broad coverage of skin tones. Images are standardized, noise is removed, and critical features are preserved for accurate classification. To address data imbalance, particularly for underrepresented skin tones, a GAN with spectral normalization generates synthetic images of dark skin lesions, enriching the dataset and enhancing model generalization. The preprocessed and augmented dataset is used to train three CNN architectures are ResNet50, NASNet, and InceptionResNetV2, each fine-tuned to optimize performance. Class weights handle imbalances, and an ensemble approach improves classification accuracy, particularly in detecting challenging conditions like melanoma across all skin tones. This combined approach of GAN augmentation and CNN-based classification offers a promising path toward more equitable dermatological diagnostics.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Skin conditions represent a significant health concern affecting individuals worldwide, with implications ranging from physical discomfort to potential life-threatening consequences. However, there exists a concerning disparity in healthcare outcomes, particularly for individuals with darker skin tones. Biases in dermatological diagnosis can lead to misdiagnosis, delayed treatment, and reduced access to appropriate care for these individuals, exacerbating existing healthcare disparities. The underrepresentation of dark skin tones in dermatological datasets and training materials is a major contributor to this issue.

Machine learning models, particularly Convolutional Neural Networks (CNNs), have become standard in dermatological image classification. However, these models may struggle to generalize effectively across diverse skin tones due to the lack of diverse representation in training datasets. This can result in lower diagnostic accuracy for individuals with darker skin. To address this, innovative approaches that account for the diversity of skin tones are necessary. The project proposes a solution that leverages both deep learning and data augmentation techniques to mitigate biases in dermatological diagnosis. Generative Adversarial Networks (GANs) are employed to generate synthetic images of dermatological conditions on dark skin tones, enriching the dataset and addressing the underrepresentation of these skin tones. By augmenting the dataset with synthetic images, we aim to improve model performance and fairness.

In addition to GAN augmentation, this project utilizes CNN architectures like ResNet, NASNet, and InceptionResNetV2 for classification. These CNN models, trained on the augmented dataset, are fine-tuned to ensure better generalization across diverse skin tones. The ensemble of these models further enhances the accuracy and fairness of the classification, balancing performance across all skin tones.

Through this combined approach, the project aims to improve the accuracy and fairness of dermatological diagnoses for individuals with darker skin tones. By reducing biases, it seeks to advance healthcare equity and accessibility, ultimately

contributing to better healthcare outcomes for marginalized communities. The following sections cover the methodology, including data collection, GAN augmentation, and CNN implementation, alongside the results and the broader implications for healthcare equity and future research.

1.1 Background

Dermatological conditions affect individuals across diverse ethnic backgrounds and skin tones. However, existing diagnostic methods often exhibit biases, leading to disparities in healthcare outcomes. Traditional approaches to dermatological diagnosis rely heavily on the availability of comprehensive datasets, which may inadequately represent certain demographic groups, particularly individuals with darker skin tones. This underrepresentation can result in misdiagnosis, delayed treatment, or suboptimal care for these populations.

The advent of deep learning techniques has opened new avenues for addressing these biases. Machine learning models have demonstrated remarkable performance in image classification tasks, including dermatological diagnosis. Deep learning techniques, particularly Convolutional Neural Networks (CNNs), have shown impressive performance in dermatological image classification. However, their effectiveness depends on diverse training data. When trained on imbalanced datasets, CNNs can inherit biases, affecting diagnostic accuracy across skin tones.

Generative Adversarial Networks (GANs) present a solution by generating synthetic images, enriching dermatological datasets with diverse examples. Combined with CNN models, this approach enhances both the fairness and accuracy of skin condition classification, addressing the limitations of underrepresented data. They hold significant promise in mitigating these biases. GANs are capable of generating synthetic data that resembles the characteristics of real-world samples. By leveraging GANs for data augmentation, it becomes possible to enrich existing dermatological datasets with diverse, representative examples, thereby reducing the impact of biased or limited training data on classification models.

1.2 Motivation

The motivation behind this research stems from the pressing need to address the biases and disparities prevalent in dermatological diagnosis. Despite advancements in medical imaging and AI-assisted diagnosis, the underrepresentation of diverse skin tones in datasets remains a significant challenge. This issue not only affects the accuracy of diagnostic models but also perpetuates healthcare inequities, potentially exacerbating the burden on marginalized communities.

By harnessing the power of GAN-based data augmentation, this research aims to bridge the gap in dataset representation, enabling the development of more robust and unbiased diagnostic models. Augmenting existing dermatological datasets with synthetic images of diverse skin tones can enhance the training process, improve generalization capabilities, and ultimately lead to more accurate and equitable diagnoses. CNNs can be decisive in skin lesion classification and can assist professionals with a portable, low-effort first diagnosis.

Moreover, the integration of advanced preprocessing techniques, such as hair removal from dermatological images, can further enhance the quality and clarity of the data, leading to improved model performance and interpretability. By addressing these challenges, this research endeavors to contribute to the broader goal of promoting healthcare equity and accessibility for all individuals, regardless of their demographic background.

1.3 Significance of Research

The significance of this research lies in its potential to address longstanding biases and disparities in dermatological diagnosis, leading to improved healthcare equity and accessibility. By leveraging GAN-based data augmentation and advanced preprocessing techniques, this study aims to develop more robust and unbiased diagnostic models, capable of accurately diagnosing skin conditions across diverse populations. Post augmentation, the CNN performs better on a diverse set of skin lesions.

Moreover, the integration of interpretability methods can provide valuable insights into the decision-making process of these models, enabling the identification and mitigation of potential biases. This transparency can foster trust in machine learning diagnosis systems and facilitate their responsible deployment in real-world healthcare settings. Additionally, this research holds significance for the broader field of medical imaging and healthcare solutions. The methodologies and frameworks developed in this study can serve as a foundation for addressing biases and underrepresentation in other medical domains, ultimately contributing to the overarching goal of equitable and inclusive healthcare for all.

1.4 Scope and Limitations

The scope of this research encompasses developing a GAN-based data augmentation framework specifically tailored for dermatological image synthesis, with an emphasis on generating diverse and representative examples. It also involves implementing advanced preprocessing techniques, including hair removal algorithms, to enhance the quality and interpretability of dermatological images. Additionally, the research aims to address biases and disparities in dermatological diagnosis, with a focus on improving healthcare outcomes for underrepresented populations. However, it is important to acknowledge the limitations of this research. The effectiveness of the proposed approach may be influenced by the quality and diversity of the initial dermatological dataset, as well as the performance of the GAN model in generating realistic and representative synthetic images. Preprocessing techniques, such as hair removal algorithms, may have limitations in handling complex or irregular hair patterns, potentially impacting the quality of the preprocessed images. Furthermore, the generalizability of the findings may be limited to the specific domain of dermatological diagnosis, and further research may be required to extend the proposed methodologies to other medical imaging applications. Moreover, while CNNs have shown great promise in skin lesion classification, their performance may vary based on the training data. CNNs can misclassify images on certain instances and this reinforces the need for a trained professional. Despite these limitations, this research endeavors to make a significant contribution to the field of dermatological diagnosis and the broader pursuit of healthcare equity and accessibility.

CHAPTER 2 RELATED WORK

2.1 Overview of Related Papers

Ian Goodfellow et al proposed the concept of Generative Adversarial Networks (GANs), which transformed generative modeling in machine learning [1]. The paper proposed a framework with two neural networks like a generator and a discriminator, engaged in a minimax game, where the generator learns to create realistic data by fooling the discriminator, while the discriminator tries to distinguish between real and fake data. GANs are praised for generating highly realistic samples without needing explicit probability models, but they also face challenges like training instability, mode collapse, and high computational requirements.

Al-Rasheed et al utilized conditional GANs (cGANs) in their research, incorporating diverse image augmentation techniques like flipping, affine transformations, contrast adjustment, Gaussian blur, and multiplication [2]. The cGAN framework trains a generator and discriminator simultaneously, with the generator learning to produce data conditioned on specific inputs while the discriminator distinguishes real from generated data, enhancing the model's ability to generate realistic, context-aware outputs. Ensemble algorithms were also applied to combine predictions from multiple models, improving classification accuracy and reducing bias. While cGANs allow for more controlled data generation and improved accuracy through ensemble methods, they face similar challenges to traditional GANs, including training instability and mode collapse.

The research done by H. Rashid et al focuses on the application of GANs for generating realistic dermoscopic images to augment training datasets for skin lesion classification [3]. The approach leverages the generative capabilities of GANs to produce diverse and representative synthetic data, offering significant advantages over conventional data augmentation methods. Studies have shown substantial improvements in evaluation metrics such as precision, recall, and F1-score in skin lesion classification tasks. However, like other GAN-based models, this approach also faces challenges such as training instability, the potential for mode collapse, and the computational intensity required for generating high-quality synthetic images.

A study by Alsaidi et al addressed the problem of class imbalance in dermoscopic image classification by comparing traditional CNN-based classification with a model trained on a GAN-augmented dataset [4]. The study evaluated the effectiveness of using data augmentation and GAN-generated synthetic data to improve classification accuracy and mitigate class imbalance in medical imaging. This study highlighted advantages, such as improving classification accuracy by enhancing underrepresented classes through GAN-augmented data. This approach helps alleviate class imbalance, a common challenge in medical image datasets, and improves the model's generalization ability. However, the study also identified disadvantages, including the complexity of training GANs and the potential for generating low-quality synthetic data, which may negatively affect classification if the discriminator is not well-tuned. Additionally, the method is computationally expensive, adding significant overhead compared to traditional CNN-based methods.

Qichen Su et al introduced the Self-Transfer GAN (STGAN) for data augmentation in skin lesion classification, addressing issues of limited data and class imbalance [5]. STGAN improves classification performance by integrating synthetic images generated by the GAN model with real images, enabling more robust training of classifiers. This approach has proven effective in generating high-quality images, crucial for medical imaging tasks. The advantage of this is that the STGAN offers a solution to data scarcity, particularly in medical applications, and helps balance class distributions, improving model performance and generalization. However, it is computationally intensive, requires careful tuning for stable training, and may not fully capture real-world data diversity, potentially affecting its overall effectiveness.

A study by E. Gören et al has explored various convolutional neural network (CNN) architectures, including CNN, MobileNet, and ResNet-18, for skin lesion classification and feature extraction [6]. These studies proposed the use of hybrid datasets that combine synthetic images generated by GAN models with original datasets, resulting in promising improvements in classification accuracy. The research also compared unconditional and conditional GANs, highlighting the importance of addressing bias in both data and models. While leveraging hybrid

datasets enhances classification accuracy and provides a robust approach to feature extraction, it risks exacerbating existing biases in the models, as GANs can perpetuate or amplify these biases. Additionally, the integration of synthetic data requires extensive computational resources and careful tuning, adding complexity to the implementation.

Shubham Innani et al proposed the use of E-GAN (Efficient-GAN) and M-GAN (Mobile-GAN) for data augmentation [7]. EGAN introduces a patch GAN-based discriminator that enhances efficiency by discerning between synthetic and real labels, optimizing the process without compromising on data quality. MGAN, on the other hand, is tailored for real-time applications, such as dermatoscopy machines, and is designed with a lightweight architecture suited for limited computational resources. The study demonstrates a deep understanding of domain-specific challenges: EGAN focuses on generating high-quality data, while MGAN emphasizes real-time efficiency for deployment in resource-constrained settings. While both models offer innovative solutions, their advantages and limitations are notable. EGAN and MGAN are highly efficient and versatile, but like other GAN models, they still face challenges such as potential training instability and the need for fine-tuning to avoid mode collapse.

Bevan and Atapour-Abarghoei's paper introduced a novel skin tone labelling algorithm aimed at improving dataset annotation, which is crucial for reducing biases in dermatological diagnosis, particularly when utilizing GAN augmentation techniques [8]. The paper explores debiasing methodologies such as 'Learning Not To Learn' and 'Turning a Blind Eye,' showcasing their efficacy in mitigating skin tone bias in skin lesion classification. These approaches, when combined with GAN augmentation, hold great promise for achieving more accurate and equitable diagnostic outcomes. By addressing biases in both dataset annotation and classification, the research significantly advances the field of fair and accurate dermatological diagnosis. The advantage of this approach is its potential to reduce bias in medical AI applications, leading to more inclusive and fairer diagnostic models. However, the method may also introduce new challenges, such as the complexity of implementing these debiasing techniques across diverse datasets,

and the risk of inadvertently obscuring important features in the data while trying to eliminate bias.

Mikołajczyk et al highlighted the importance of evaluation metrics in assessing the performance of GANs in mitigating biases [9]. The study emphasized metrics such as fidelity, diversity, training speed, and classifier performance on datasets composed of both real and synthetic data. Across various methods, models trained on GAN-augmented datasets demonstrated substantial improvements in performance. This approach is advantageous for generating balanced datasets, reducing bias, and enhancing the accuracy of machine learning models. However, the use of GANs in this context can also come with drawbacks such as increased complexity in evaluation, the risk of generating synthetic data that may still contain latent biases, and the need for extensive computational resources for training.

The paper by Alankrita Aggarwal et al highlights the use of Convolutional Neural Networks (CNNs) for image segmentation, emphasizing their role in feature extraction and pixel-level classification, which significantly improves performance in applications like medical image analysis and traffic management [10]. The integration of CNNs with Generative Adversarial Networks (GANs) further refines the synthesis of realistic images, enhancing segmentation methodologies. CNNs, when combined with GANs, excel at capturing intricate patterns and creating high-quality segmentation results, but they also present challenges, such as computational complexity, potential training instability, and the need for large annotated datasets to achieve optimal performance.

A study by Sara Atito Ali Ahmed et al on skin lesion classification employed advanced Convolutional Neural Networks (CNNs), particularly Xception, Inception-ResNet-V2, and NasNetLarge architectures, within the ISIC2019 framework to enhance skin cancer detection and classification [11]. They addressed key challenges such as class imbalance and anomaly detection by integrating ensemble techniques like LightGBM and one-class classification methods. The study demonstrated that these techniques improved diagnostic accuracy. While CNNs and ensemble methods offer significant advantages by boosting performance in medical imaging, the approach still faces limitations, such as the

complexity of training deep learning models, the need for large datasets, and potential overfitting in cases of small or imbalanced data.

The paper by Gouda et al highlights the use of deep learning techniques, particularly convolutional neural networks (CNNs), in automating early skin cancer detection [12]. By utilizing large datasets like ISIC, these models demonstrate high accuracy in classifying skin lesions, including melanoma. Innovations such as data augmentation and super-resolution GANs further enhance the performance of these models by providing clearer input data. These advancements improve the diagnostic process by addressing the limitations of manual assessments by dermatologists, offering speed and reliability. However, while CNN-based models achieve high accuracy, they also face challenges like the need for large labeled datasets, potential biases in training data, and the risk of overfitting when handling diverse skin types and lesions.

2.2 Research Gap

Despite advancements in machine learning and generative models like GANs, a notable gap persists in the classification of skin lesions on darker skin tones. A detailed analysis of studies reveals that dermatological models, particularly those trained on imbalanced datasets, often fail to generalize well across diverse skin tones, leading to biases in diagnostic outcomes.

2.2.1 Limited focus on skin tone diversity in GAN-based models

While several studies utilize GANs for image augmentation in dermatology, there is a notable lack of research specifically targeting the generation of synthetic images that accurately represent diverse skin tones. Hence, the project aims to address this gap by focusing on underrepresented skin types, which is crucial for improving the fairness of diagnostic models.

2.2.2 Bias mitigation in dermatological diagnosis

Although some studies, like Bevan and Atapour-Abarghoei's work, explore bias reduction techniques, there is still a need for a comprehensive framework that directly tackles bias mitigation within GAN-augmented datasets. Most works focus

on enhancing classification accuracy but lack a targeted approach for addressing healthcare equity, particularly for marginalized populations.

2.2.3 Advanced preprocessing techniques, including hair removal

Existing GAN-based studies for dermatological image augmentation do not prioritize advanced preprocessing steps like hair removal, which is vital for improving image clarity and interpretability. Thus, the project addresses this gap by incorporating preprocessing methods that enhance the quality of the generated images.

2.2.4 Class-specific lesion diagnosis (Melanoma, SCC, BCC)

Most research focuses on binary classification (benign vs. malignant) without delving deeply into differentiating between specific malignant types like melanoma, squamous cell carcinoma, and basal cell carcinoma. So, the project's objective is to classify these specific cases represents an important advancement in granularity and diagnostic precision.

2.2.5 GANs for healthcare equity

While GANs have been extensively used for data augmentation, few projects align this technique with the goal of promoting healthcare equity. The project explicitly focus on mitigating bias in dermatological diagnosis for underrepresented populations marks a novel contribution to the field. This gap highlights the need for both technical advancements and an ethical focus in medical image analysis.

CHAPTER 3 PROBLEM STATEMENT AND REQUIREMENTS SPECIFICATION

3.1 Problem Statement

The under-representation of certain skin-types and ethnicities can lead to misdiagnoses and healthcare disparities. This bias can perpetuate health inequalities, as certain demographic groups may receive less accurate diagnoses. Using a GAN-based augmentation strategy, this project aims to improve dataset inclusivity and enhance the generalization of classification models. Following augmentation, CNNs are employed to classify skin lesions, providing a more equitable and robust diagnostic approach across skin types. Additionally, the project focuses on improving the overall accuracy and performance of the models, ensuring better healthcare outcomes for all individuals.

3.2 Objectives

The primary objectives of this project are as follows:

- To develop a GAN-based data augmentation framework tailored for dermatological image synthesis, with a focus on generating synthetic images that accurately represent diverse skin tones and lesion characteristics.
- To implement advanced preprocessing techniques, including hair removal algorithms, to enhance the quality and interpretability of dermatological images, facilitating more accurate diagnosis.
- To classify skin lesions into benign and malignant cases and further into melanoma, squamous cell carcinoma (SCC), and basal cell carcinoma (BCC) if found to be malignant.
- To contribute to the advancement of healthcare equity by providing a framework for mitigating biases in dermatological diagnosis, ultimately improving healthcare outcomes for underrepresented populations.

3.3 Software Requirements

3.3.1 Torch and TensorFlow

Torch (or PyTorch) and TensorFlow are two of the most widely used frameworks for implementing deep learning models, including Generative Adversarial Networks (GANs). Both frameworks provide extensive libraries and tools for developing, training, and deploying neural networks. PyTorch is known for its dynamic computational graph, which allows for flexible model building and easier debugging, making it particularly popular among researchers and developers. TensorFlow, on the other hand, offers robust deployment capabilities and is favored for production environments. Both frameworks support GPU acceleration, enabling efficient training of complex models on large datasets, making them ideal choices for the implementation of GAN augmentation and classification models.

3.3.2 Torch-GAN

Torch-GAN is a specialized library built on top of PyTorch that simplifies the process of constructing and training Generative Adversarial Networks. It provides pre-built modules and functions specifically designed for GAN development, such as various loss functions, training loops, and architectural components tailored for different GAN variants. This library facilitates the experimentation with different GAN configurations, allowing researchers and developers to focus more on the creative aspects of model design rather than the underlying implementation details. By leveraging Torch-GAN, users can efficiently build, train, and evaluate their GAN models while ensuring high performance and ease of use.

3.3.3 OpenCV or PIL

OpenCV (Open-Source Computer Vision Library) and PIL (Python Imaging Library, now maintained as Pillow) are powerful libraries used for image preprocessing and manipulation. OpenCV is particularly well-suited for real-time computer vision tasks and provides extensive functionality for image processing, including filtering, transformation, and feature extraction. It is widely used in applications that require robust image manipulation capabilities. On the other hand, PIL offers a user-friendly interface for common image tasks such as opening, manipulating, and saving image files. Both libraries play a crucial role in preparing images for training

machine learning models, ensuring that the data is in the appropriate format and quality for effective model learning.

3.3.4 Anaconda

Anaconda is a distribution of the Python and R programming languages designed for scientific computing and data analysis. It simplifies package management and deployment, making it easy to install, update, and manage libraries and dependencies required for various projects. Anaconda comes with its own package manager, conda, which allows users to create isolated environments for different projects, ensuring that dependencies do not conflict with each other. This feature is particularly beneficial when working with multiple projects that may require different versions of libraries or Python itself. By using Anaconda, researchers can maintain a clean and efficient development environment, facilitating smoother workflow and collaboration.

3.3.5 Pandas

The Pandas library is a powerful tool for data manipulation and analysis in Python. It provides data structures, such as Data Frames, that allow for efficient handling of large datasets, enabling operations like filtering, grouping, merging, and reshaping data with ease. In the context of machine learning, Pandas is invaluable for data preprocessing, allowing researchers to clean and prepare datasets before feeding them into models. Its intuitive syntax and functionality enable users to perform complex data operations quickly, making it a preferred choice for handling datasets in various applications, including those in healthcare and dermatology.

3.3.6 Jupyter Notebook

Jupyter Notebook is an integrated development environment (IDE) that provides an interactive interface for coding, data analysis, and visualization. It allows users to create documents that combine live code, equations, visualizations, and narrative text, making it ideal for experimentation and iterative development. Jupyter supports multiple programming languages, including Python, and is particularly popular in data science and machine learning communities. The ability to execute code in small, manageable chunks while visualizing results in real-time

makes it an excellent tool for conducting experiments, documenting findings, and sharing results with collaborators or stakeholders.

3.3.7 Git

Git is a distributed version control system that enables developers to track changes in their code and collaborate effectively on projects. Git's ability to maintain a complete history of project modifications makes it an invaluable tool for managing code over time. Additionally, platforms like GitHub and GitLab enhance Git's capabilities by providing remote repositories and collaborative features, enabling teams to share code, conduct code reviews, and manage issues. By incorporating Git into the development workflow, teams can ensure better organization, collaboration, and quality control throughout the project lifecycle.

3.4 Hardware Requirements

3.4.1 CPU

Minimum Intel Core i5 or AMD Ryzen 5 processor:

A good quality CPU (central processing unit) is essential for handling computations efficiently. Intel Core i5 or AMD Ryzen 5 processors provide sufficient processing power for running deep learning models and handling data-intensive tasks without significant delays.

3.4.2 GPU

NVIDIA GeForce GTX 1060 or higher:

A dedicated GPU (graphics processing unit) is important for accelerating the training of machine learning models, especially those that involve large datasets and complex computations. The GTX 1060 or a better model helps in performing parallel processing, which speeds up tasks like image processing and neural network training.

3.4.3 RAM

Minimum 16GB:

Having at least 16GB of RAM (random access memory) is crucial for multitasking and managing large datasets during processing. Sufficient RAM ensures that your

computer can run multiple applications simultaneously without slowing down, which is particularly important when training models or processing images.

3.4.4 Storage

SSD storage with sufficient capacity (at least 256GB):

An SSD (solid-state drive) is recommended for faster data access and loading times compared to traditional hard drives. A minimum capacity of 256GB is important to store the operating system, software, and datasets efficiently. The speed of an SSD helps in reducing the time taken to read and write data, which is beneficial during model training and data manipulation tasks.

3.5 CNN Architectures used in Deep Learning

3.5.1 ResNet50

ResNet50 is a deep residual network that consists of 50 layers, designed to address the vanishing gradient problem that can occur in very deep networks. Introduced by Kaiming He et al. in their paper "Deep Residual Learning for Image Recognition," ResNet employs skip connections, or residual connections, that allow gradients to flow more easily through the network during training. These connections enable the model to learn identity mappings, which help preserve information and facilitate the training of deeper networks. ResNet50 is particularly effective for image classification tasks and has been widely adopted in applications ranging from medical imaging to object detection. Its architecture allows it to achieve high accuracy on benchmark datasets like ImageNet while maintaining relatively low computational requirements.

3.5.2 NASNet

NASNet (Neural Architecture Search Network) is a model architecture that was designed using neural architecture search techniques. Developed by Barret Zoph et al., NASNet optimizes the architecture of the CNN through a search algorithm that explores various configurations to find the best-performing model for specific tasks. The NASNet architecture is modular, consisting of cells that can be repeated to create a deeper network. This model has shown to achieve state-of-the-art results on various image classification benchmarks, thanks to its ability to balance

complexity and performance. NASNet is particularly noted for its scalability, meaning that it can be adapted for different computational budgets and performance requirements, making it suitable for both mobile and server-based applications.

3.5.3 InceptionResNetV2

InceptionResNetV2 combines the Inception architecture with residual connections, creating a hybrid model that leverages the strengths of both approaches. The Inception architecture, initially introduced in the GoogLeNet model, utilizes multiple filter sizes in parallel within the same layer to capture various features from the input image. This approach enhances the model's ability to learn from different spatial resolutions. By incorporating residual connections, InceptionResNetV2 addresses the challenges associated with training very deep networks, similar to ResNet. The combination of Inception modules and residual connections allows InceptionResNetV2 to achieve high accuracy on image classification tasks while being efficient in terms of computational resources. It has been widely used in various applications, including fine-grained image classification and transfer learning tasks.

CHAPTER 4 METHODOLOGY AND DESIGN

4.1 Methodology

The proposed system integrates both GAN-based data augmentation and CNN-based classification models to improve dermatological diagnosis across diverse skin tones. The system follows these steps:

4.1.1 Data Collection

Gather a diverse dermatological dataset containing images of various skin conditions across different demographics and skin types. Ensure the dataset is labeled for classification (e.g., benign or malignant skin lesions). Thid project makes use of the ISIC 2019 dataset.

4.1.2 Preprocessing

Preprocess the images by resizing them to a uniform size (e.g., 224x224 pixels) to ensure consistency. Normalize the pixel values, and apply basic augmentation techniques such as rotation, flipping, and cropping to increase the dataset's variability. Additionally, use techniques like hair removal for clearer images.

4.1.3 GAN Training

Train a Generative Adversarial Network (GAN) to generate synthetic dermatological images. The GAN comprises two components: the generator, which creates synthetic images, and the discriminator, which distinguishes between real and synthetic images. The GAN is trained until it can generate realistic images of skin lesions that mimic real samples.

4.1.4 GAN Augmentation

Once the GAN is trained, use the generator to create a diverse set of synthetic dermatological images, especially focusing on underrepresented skin tones. The quantity of generated images can be controlled through parameters like noise input and batch size.

4.1.5 Dataset Expansion

Combine the original dataset with the synthetic images generated by the GAN. This augmented dataset provides more comprehensive representation across different skin tones, helping to reduce bias in training. It is important to maintain the balance between real and synthetic images to ensure the model generalizes well.

4.1.6 CNN Model Training

Train Convolutional Neural Networks (CNNs) on the expanded dataset. In this project, advanced CNN architectures such as ResNet, InceptionResNetV2, and NASNet are employed. These models are well-suited for image classification tasks, with the ability to extract deep features from images. Each CNN model is fine-tuned to improve performance on the augmented dataset.

4.1.7 Model Evaluation

Evaluate the performance of both the GAN and the CNN models. The GAN is assessed by how realistic the generated images are, while the CNN models are evaluated using metrics to determine their effectiveness in classifying skin conditions. The classification results are compared before and after GAN augmentation to highlight improvements.

4.1.8 Iterative Improvement

Analyze the performance of both the GAN and CNN models and make iterative improvements. This could involve fine-tuning hyperparameters, adjusting augmentation techniques, or optimizing the network architecture. The goal is to maximize accuracy and generalization.

4.1.9 Cross - validation

To check for overfitting, the model is validated and tested on unseen, labeled images. This helps assess its ability to generalize beyond the training data. This determines how the models perform in a real-life scenario.

4.1.10 Deployment

Finally, deploy the trained models for real-world applications in dermatological diagnosis. This involves integrating the models into a user-friendly interface or a healthcare system, ensuring privacy, security, and regulatory compliance.

4.2 System Architecture

4.2.1 GAN

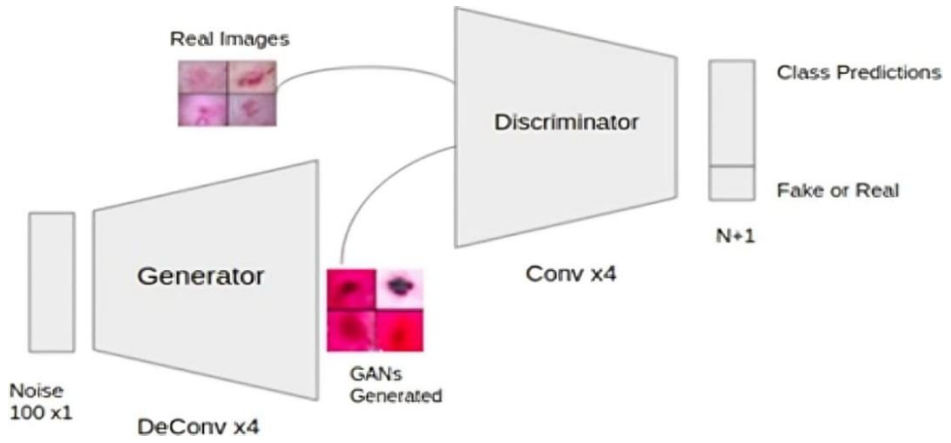


Fig.4.1 Architecture of Generative Adversarial Network (GAN) consisting of generator and discriminator

Fig 4.1 of the Spectral Normalization GAN (SN-GAN) used in this project consists of two key components: the Generator and the Discriminator. The Generator receives a noise vector sampled from a Gaussian distribution as input and passes it through several layers to synthesize high-quality dermatological images. It starts with a Dense (fully connected) layer, which reshapes the noise into an initial feature map, followed by transposed convolutional layers that progressively upscale the image to the target resolution of 224x224. These layers are accompanied by batch normalization to stabilize the learning process and ReLU activation functions to introduce non-linearity. The final layer uses the tanh activation to scale the pixel values between -1 and 1.

The Discriminator, which aims to distinguish real from synthetic images, takes an image as input and processes it through a series of convolutional layers that downsample the image while extracting meaningful features. To prevent overfitting and enhance stability during training, Spectral Normalization is applied to each

layer, ensuring controlled gradients. LeakyReLU activation is employed after each layer to avoid dying neurons. The network also includes dropout layers to improve generalization. The output of the Discriminator is a single scalar, produced by a sigmoid activation, representing the probability that the input image is real.

For the loss functions, the GAN employs the binary cross-entropy loss for both the Generator and Discriminator. The Generator's loss is designed to minimize the ability of the Discriminator to differentiate between real and generated images, while the Discriminator's loss aims to maximize its ability to distinguish real images from the synthetic ones. The architecture and spectral normalization together help maintain balanced training between the two networks, reducing the risk of mode collapse and ensuring high-quality image generation.

4.2.2 CNN

The project makes use of three Convolutional Neural Networks (CNNs) — ResNet50, NASNet, and InceptionResNetV2 — were employed for skin lesion classification. Each model was chosen for its unique architecture and strengths, ensuring comprehensive feature extraction and enhanced diagnostic accuracy across diverse skin tones.

The ResNet50 model is a deep convolutional neural network with 50 layers. It introduces residual connections that help mitigate the vanishing gradient problem by allowing gradients to bypass multiple layers. This is achieved through identity mappings, which connect earlier layers directly to later ones, ensuring smoother flow of information. ResNet50 begins with a convolutional layer followed by a max pooling layer, then several residual blocks consisting of convolutional layers with batch normalization and ReLU activations. These blocks are designed to efficiently extract features while maintaining computational efficiency. The final layers consist of a global average pooling layer and a fully connected layer with a softmax activation, outputting the probabilities for each class. The model is optimized using categorical cross-entropy loss.

The NASNet model is a neural architecture search (NAS)-based network, automatically optimized for performance through reinforcement learning. NASNet

consists of normal cells and reduction cells that alternate in the network, where normal cells maintain the spatial dimensions of the input, and reduction cells downsample the image. Each cell contains multiple layers, including convolutions, batch normalization, and ReLU activations, all arranged in a highly optimized manner. The architecture ends with a global average pooling layer followed by a fully connected layer and a softmax activation for multi-class classification. NASNet's architecture is particularly powerful in adapting to different datasets, offering high accuracy with minimal manual intervention. It also uses categorical cross-entropy loss for optimization.

The InceptionResNetV2 model combines the Inception architecture with residual connections, enhancing both computational efficiency and performance. The network is built using Inception blocks, which apply multiple filters of varying sizes (1x1, 3x3, and 5x5) in parallel to capture different features at different scales, followed by concatenation to merge these feature maps. The residual connections between the Inception blocks ensure stable gradient flow, preventing degradation in deep networks. The architecture begins with standard convolutional layers, followed by multiple stacked Inception and residual blocks, and concludes with global average pooling and a fully connected layer for classification. Softmax activation is used for the final output, and the model is optimized using categorical cross-entropy loss.

4.3 Data Flow Diagram

The Fig 4.2 outlines the systematic approach taken in the project, beginning with project planning and research, which establishes the foundation for subsequent steps. Following this, the data collection and preprocessing stage ensures that high-quality data is gathered and prepared for analysis. This leads to the GAN model development for augmentation, where Generative Adversarial Networks (GANs) are created to enhance the dataset by generating synthetic images, particularly targeting the underrepresentation of darker skin tones in dermatological datasets. Once the GAN model is developed, it moves into the GAN model training and optimization phase, where the model is refined for better performance in generating useful images. This training phase is crucial for producing high-quality synthetic data that can enhance the overall dataset.

Simultaneously, there's a focus on the preparation of the dataset for CNN classification, ensuring that the data is well-structured for the next phases.

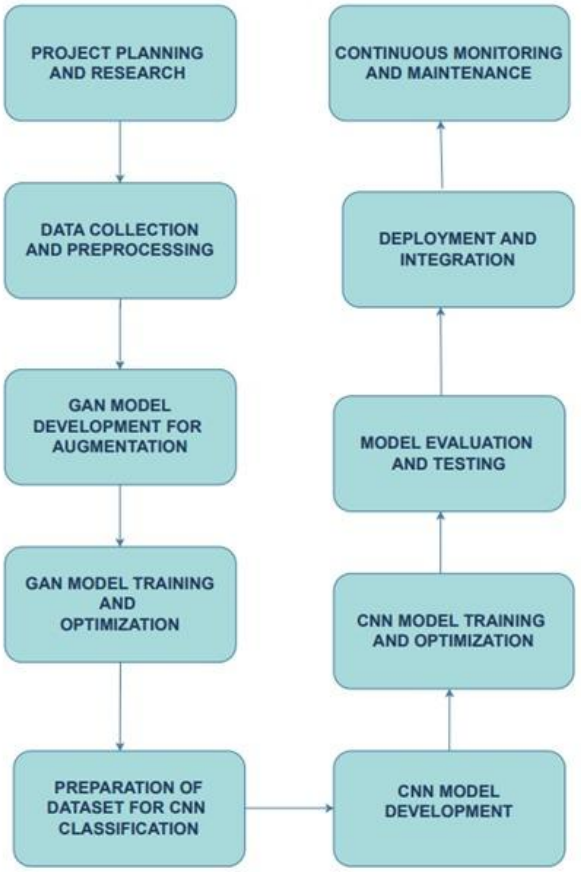


Fig.4.2 Data Flow Diagram

Following the GAN model training, the flowchart branches into two critical paths: CNN model training and optimization, where Convolutional Neural Networks (CNNs) are trained using both real and synthetic images, and CNN model development, which involves building the CNN architecture tailored for the task. After these models are optimized, the project culminates in model evaluation and testing, ensuring that the performance meets the required standards before proceeding to deployment and integration. Finally, the project emphasizes continuous monitoring and maintenance, ensuring that the system remains effective and reliable post-deployment. This comprehensive approach fosters a robust development cycle that aims to improve dermatological diagnoses across diverse skin tones.

4.4 Summary

The system design outlines the architecture for the Spectral Normalization GAN (SN-GAN) and three Convolutional Neural Networks (CNNs) utilized for dermatological image classification. The SN-GAN consists of a Generator that synthesizes high-quality dermatological images from noise vectors and a Discriminator that distinguishes between real and synthetic images, with both components using binary cross-entropy loss for balanced training. The CNNs like ResNet50, NASNet, and InceptionResNetV2, were selected for their distinct architectural advantages, enabling effective feature extraction and accurate skin lesion classification across varying skin tones. ResNet50 employs residual connections to address gradient issues, NASNet utilizes a reinforcement learning-based architecture for optimization, and InceptionResNetV2 combines Inception blocks with residual connections for enhanced performance. Additionally, a data flow diagram illustrates the systematic approach taken in the project, highlighting the stages from initial research and data collection to GAN model development, training, and CNN optimization. The process culminates in model evaluation and deployment, followed by continuous monitoring, thereby aiming to improve dermatological diagnoses and address biases related to skin tone representation.

CHAPTER 5 SYSTEM IMPLEMENTATION

5.1 Preprocessing

In Fig 5.1, code snippet imports libraries for deep learning (PyTorch), computer vision (OpenCV), and related utilities (datasets, transformations, optimization algorithms, tensor-image conversion, and operating system interactions).

```
# Import necessary libraries
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torchvision.utils import save_image
import cv2
import os
```

Fig 5.1 Importing libraries

5.1.1 Resizing Images

```
folder_path = "images"
target_size = (224, 224)
```

```
def resize_images_in_folder(folder_path, target_size):
    # Create a new folder to store resized images
    resized_folder_path = os.path.join(folder_path, "resized")
    os.makedirs(resized_folder_path, exist_ok=True)

    # Iterate through each image
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        # Check if the file is an image (JPEG or PNG)
        if os.path.isfile(file_path) and (file_name.lower().endswith('.jpg')
            # Load and resize the image
            image = cv2.imread(file_path)
            resized_image = cv2.resize(image, target_size)

            # Save the resized image to the new folder
            resized_image_path = os.path.join(resized_folder_path, file_name)
            cv2.imwrite(resized_image_path, resized_image)
            # print(f"Resized image saved: {resized_image_path}")
```

```
resize_images_in_folder(folder_path, target_size)

print("Resizing complete!")
```

Fig 5.2 Resizing images

In Fig 5.2 code snippet defines a folder_path and target_size for resizing images. The resize_images_in_folder function creates a new "resized" folder, then iterates through each file in the original folder. For each JPEG or PNG image, it loads the image using OpenCV's cv2.imread, resizes it to the target_size using cv2.resize, and saves the resized image to the "resized" folder using cv2.imwrite. Finally, the function is called, and a message is printed when the resizing is complete.

This code can be useful for preprocessing image data in machine learning tasks, where it's often necessary to resize images to a specific size required by the neural network architecture or for other image processing tasks.

5.1.2 Darkening Images

```python
import random
# Function to generate darker images
def generate_darker_images(input_folder, output_folder, factor, apply_probability=0.25):
    # Create output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Loop through images in the input folder
    for filename in os.listdir(input_folder):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            # Read image
            img_path = os.path.join(input_folder, filename)
            img = cv2.imread(img_path)

            # Randomly decide whether to apply darker effect
            if random.random() < apply_probability:
                # Apply darker effect
                darker_img = cv2.convertScaleAbs(img, alpha=factor, beta=0)
            else:
                # No change, keep the original image
                darker_img = img.copy()

            # Save image
            output_path = os.path.join(output_folder, filename)
            cv2.imwrite(output_path, darker_img)

# Example usage
input_folder = "input_images_folder"
output_folder = "output_images_folder"
darkening_factor = 0.5  # Adjust this factor as needed, lower values make the image darker

generate_darker_images('images/resized', 'darkened', darkening_factor)
```

Fig 5.3 Darkening images

Fig 5.3 code snippet defines a function generate_darker_images that takes an input folder of images, an output folder, a darkening factor, and an optional probability for applying the effect. It creates the output folder if it doesn't exist, then loops through each image file in the input folder. For each image, it randomly decides whether to apply the darkening effect based on the provided probability. If applied, it uses OpenCV to scale the pixel values of the image by the specified darkening factor, making the image darker. Otherwise, it keeps the original image unchanged. Finally, it saves the processed (darkened or unchanged) image to the output folder.

The code also includes an example usage, specifying the input folder, output folder, and darkening factor. This function can be useful for data augmentation in machine learning tasks involving image data, introducing variations in brightness to improve model robustness.

5.1.3 Hair Removal from Images

```
#for directory - hair removal (quality compromise))

def process_images_with_hair(input_folder, output_folder):
    # Create output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Loop through images in the input folder
    for filename in os.listdir(input_folder):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            # Read image
            img_path = os.path.join(input_folder, filename)
            src = cv2.imread(img_path)

            # Convert the original image to grayscale
            grayScale = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

            # Kernel for the morphological filtering
            kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (17, 17))

            # Perform the blackHat filtering on the grayscale image to find the hair contours
            blackhat = cv2.morphologyEx(grayScale, cv2.MORPH_BLACKHAT, kernel)

            # Intensify the hair contours in preparation for the inpainting algorithm
            _, thresh2 = cv2.threshold(blackhat, 10, 255, cv2.THRESH_BINARY)

            # Inpaint the original image depending on the mask
            dst = cv2.inpaint(src, thresh2, 1, cv2.INPAINT_TELEA)

            # Save the processed image
            output_path = os.path.join(output_folder, filename)
            cv2.imwrite(output_path, dst)

            print(f"Processed: {filename}")

# Example usage
input_folder = "darkened"
output_folder = "finalpreprocessed"

process_images_with_hair(input_folder, output_folder)
```

Fig 5.4 Hair removal from images

Fig 5.4 code snippet defines a function process_images_with_hair that removes hair from images using OpenCV operations. It takes an input folder of images and an output folder to save the processed images. For each image in the input folder, it converts the image to grayscale, applies a BlackHat morphological operation to find hair contours, intensifies the hair contours, and then uses OpenCV's inpainting technique to remove the hair by replacing it with plausible values based on the surrounding areas. The processed image without hair is saved in the output folder. The function also prints the processed filename. An example usage is provided, specifying the input folder as "darkened" and the output folder as "finalpreprocessed". This function can be helpful for preprocessing image data in machine learning tasks involving skin or hair detection, where removing hair can improve model accuracy.

5.2 GAN Model

This Generative Adversarial Network (GAN) implementation focuses on generating high-resolution images, specifically 256x256 pixels. The architecture comprises a generator and a discriminator, utilizing techniques such as spectral normalization and label smoothing to enhance training performance.

5.2.1 Generator Network Architecture

```
# Generator for 256x256 images
class Generator_256(nn.Module):
    def __init__(self, ngpu, nz, ngf, nc):
        super(Generator_256, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            nn.ConvTranspose2d(nz, ngf * 32, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 32),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 32, ngf * 16, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 16),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 16, ngf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, input):
        return self.main(input)
```

Fig 5.5 Generator network

Fig 5.5 of Generator_256 class defines a generator model for producing 256x256 images in a Generative Adversarial Network (GAN). It inherits from nn. Module and initializes its architecture in the constructor. The model consists of a series of transposed convolutional layers, each followed by Batch Normalization and a ReLU activation function, allowing for effective upsampling of the input noise vector.

The generator begins with a transposed convolution that transforms the input latent vector (nz) into a feature map with dimensions suitable for further upsampling. Subsequent layers progressively increase the spatial dimensions while reducing the depth, eventually outputting a 256x256 image with a specified number of channels (nc). The final layer employs the Tanh activation function, ensuring that the output values are normalized between -1 and 1, suitable for image generation tasks. The model's forward method passes the input through the sequential layers, producing the generated image.

5.2.2 Discriminator Architecture Definition

```
# Discriminator with Spectral Normalization for 256x256 images
class Discriminator_SN_256(nn.Module):
    def __init__(self, ngpu, nc, ndf):
        super(Discriminator_SN_256, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            nn.utils.spectral_norm(nn.Conv2d(nc, ndf, 4, 2, 1, bias=False)),
            nn.LeakyReLU(0.2, inplace=True),
            nn.utils.spectral_norm(nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False)),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            nn.utils.spectral_norm(nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False)),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.utils.spectral_norm(nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False)),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.utils.spectral_norm(nn.Conv2d(ndf * 8, ndf * 16, 4, 2, 1, bias=False)),
            nn.BatchNorm2d(ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),
            nn.utils.spectral_norm(nn.Conv2d(ndf * 16, 1, 4, 1, 0, bias=False)),
            nn.AdaptiveAvgPool2d(1)
        )

    def forward(self, input):
        return self.main(input).view(-1)
```

Fig 5.6 Discriminator network

Fig 5.6 Discriminator_SN_256 class implements a discriminator model for 256x256 images in a Generative Adversarial Network (GAN) using spectral normalization. Inheriting from cnn. Module, it initializes its layers in the constructor.

The architecture comprises a series of convolutional layers, each employing spectral normalization to stabilize the training process by controlling the Lipschitz constant. The model begins with a convolutional layer that reduces the spatial dimensions while increasing the feature depth. This is followed by a Leaky ReLU activation function, allowing the network to learn from both positive and negative inputs.

As the input progresses through the layers, it undergoes additional convolutional transformations, batch normalization, and Leaky ReLU activations, maintaining a balanced gradient flow. The final convolutional layer produces a single output representing the probability that the input image is real, followed by an adaptive average pooling layer that compresses the feature map into a fixed-size output. The forward method reshapes the output to a one-dimensional tensor, suitable for subsequent loss calculations.

5.2.3 Weights class and loading dataset

```
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

Fig 5.7 Weights class

Fig 5.7 of weights_init function initializes the weights of the neural network layers, ensuring that convolutional layers have weights sampled from a normal distribution centered at zero with a standard deviation of 0.02, while Batch Normalization layers are initialized with weights centered at one. This initialization helps facilitate effective training by preventing saturation of the activation functions.

```python
class SingleClassDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.images = [f for f in os.listdir(root_dir) if f.endswith('.jpg') or f.endswith('.png')]

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_name = os.path.join(self.root_dir, self.images[idx])
        image = Image.open(img_name).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image, 0  # 0 is a dummy label
```

Fig 5.8 Loading the dataset

Fig 5.8 of SingleClassDataset class is defined to handle the loading of images from a specified directory, allowing for easy integration into a DataLoader. The dataset only supports one class, which is useful for training GANs focused on a specific type of image. The create_single_class_dataloader function constructs a DataLoader for this dataset, applying a series of transformations to prepare images for training, including resizing, normalization, and tensor conversion. This encapsulation of data handling simplifies the preprocessing pipeline for the GAN training process.

5.2.4 Training Loop for GAN

In the Fig 5.9 of training loop function conducts the training process for a Generative Adversarial Network (GAN) comprising a generator (netG) and a discriminator (netD). It accepts parameters for the number of epochs, data loader, device settings, loss criterion, and optimizers for both networks, among others. The training process initiates with the creation of lists to store generated images and losses for both the generator and discriminator. Labels for real and fake images are defined, incorporating an option for label smoothing to enhance training stability.

During each epoch, the loop iterates through batches of data. The discriminator is updated first by processing real images, computing the corresponding loss, and backpropagating the gradients. Subsequently, the generator produces fake images, which are evaluated by the discriminator to calculate the loss for the generator. The weights for both networks are then updated based on their respective losses. The function also logs training statistics at regular intervals and

saves generated images for visual evaluation. It concludes by returning the lists of generator and discriminator losses, along with the generated image outputs. The GAN successfully trains to produce realistic images through iterative training of the generator and discriminator. The accompanying visualization functions facilitate evaluation of the training process and the quality of generated images.

```
def training_loop(num_epochs, dataloader, netG, netD, device, criterion, nz, optimizerG, optimizerD, fixed_noise, out, use_
    img_list = []
    img_list_only = []
    G_losses = []
    D_losses = []
    iters = 0

    real_label = smooth_label if use_label_smoothing else 1.
    fake_label = 0.
    os.makedirs(os.path.dirname(out), exist_ok=True)
    print("Starting Training Loop...")
    for epoch in range(num_epochs):
        for i, data in enumerate(dataloader, 0):
            # ... (training loop code remains unchanged)
            ############################
            # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
            ###########################
            netD.zero_grad()
            real_cpu = data[0].to(device)
            b_size = real_cpu.size(0)
            label = torch.full((b_size,), real_label, dtype=torch.float, device=device)
            output = netD(real_cpu)
            errD_real = criterion(output, label)
            optimizerD.step()

            ############################
            # (2) Update G network: maximize log(D(G(z)))
            ###########################
            netG.zero_grad()
            label.fill_(real_label)  # fake labels are real for generator cost
            output = netD(fake)
            errG = criterion(output, label)
            errG.backward()
            D_G_z2 = output.mean().item()
            optimizerG.step()

            # Output training stats
            if i % 50 == 0:
                print(f'[{epoch}/{num_epochs}][{i}/{len(dataloader)}]\tLoss_D: {errD.item():.4f}\tLoss_G: {errG.item():.4f}

            # Save Losses for plotting later
            G_losses.append(errG.item())
            D_losses.append(errD.item())

            # Check how the generator is doing by saving G's output on fixed_noise
            if (iters % 500 == 0) or ((epoch == num_epochs-1) and (i == len(dataloader)-1)):
                with torch.no_grad():
                    fake = netG(fixed_noise).detach().cpu()
                img_list.append(vutils.make_grid(fake, padding=2, normalize=True))
                img_list_only.append(fake)

                for j in range(len(fake)):
                    save_image(fake[j], f"{out}{epoch}_{j}_img.png")

            iters += 1

    return G_losses, D_losses, img_list, img_list_only
```

Fig 5.9 Training loop

5.3 CNN Model

5.3.1 InceptionResNetV2 Implementation

In this section, we detail the implementation of the InceptionResNetV2 model for classifying skin lesions. The following subsections outline the data preparation, model architecture, training process, evaluation, and visualization.

5.3.1.1 Setup and Libraries

As illustrated in Fig. 5.10, we start by importing the required libraries and setting the constants for image processing and model training parameters.

```
import tensorflow as tf
from tensorflow.keras.applications import InceptionResNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

# Set up constants
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS = 50
```

Fig 5.10 Setup and Libraries

5.3.1.2 Data Preparation

In Fig 5.11, we set up the data generators to augment and preprocess images for training the InceptionResNetV2 model. Using the ImageDataGenerator class, we apply various transformations, such as rotation, width and height shifts, and zooming, which enhance the dataset's variability. The dataset is divided into four classes: Melanoma (MEL), Nevus (NV), Basal Cell Carcinoma (BCC), and Squamous Cell Carcinoma (SCC). This preparation is crucial for training a robust model that generalizes well to unseen data.

```
# Define the path to the images folder
data_dir = '/kaggle/input/images-skinlesion/images'
class_names = ['MEL', 'NV', 'BCC', 'SCC']

# Set up data generators with increased augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training',
    classes=class_names,
    shuffle=True
)

validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation',
    classes=class_names,
    shuffle=False
)
```

Fig 5.11 Data Preparation in InceptionResNetV2

5.3.1.3 Model Architecture and Model Compilation

```
# Load pre-trained InceptionResNetV2 model
base_model = InceptionResNetV2(weights='imagenet', include_top=False, input_shape=(*IMG_SIZE, 3))
```

```
Found 7174 images belonging to 4 classes.
Found 1791 images belonging to 4 classes.
```

```
# Fine-tune the model
for layer in base_model.layers:
    layer.trainable = True

# Add custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, kernel_regularizer='l2', activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(4, activation='softmax')(x)

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

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

# Compute class weights
class_weights = compute_class_weight('balanced', classes=np.unique(train_generator.classes), y=train_generator.classes)
class_weight_dict = dict(enumerate(class_weights))

# Define callbacks
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=0.00001, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=1)
model_checkpoint = ModelCheckpoint('/kaggle/working/best_model_inceptionresnetv2.keras', save_best_only=True, monitor='val_
```

Fig 5.12 Model Architecture and Model Compilation in InceptionResNetV2

Fig 5.12 illustrates the InceptionResNetV2 model architecture, utilizing pre-trained weights from the ImageNet dataset for feature extraction. We fine-tune the model

by making all layers trainable, allowing it to adapt to skin lesion features. Custom layers, such as global average pooling, dense layers, and dropout, enhance classification into four categories, with the final layer using SoftMax activation for output probabilities. As shown in Fig 5.12, the model is compiled with the Adam optimizer at a learning rate of 0.0001 and uses categorical cross-entropy as the loss function. Evaluation metrics include accuracy, which assesses the model's overall performance during training.

5.3.1.4 Training the Model and Model Evaluation

```
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    class_weight=class_weight_dict,
    callbacks=[reduce_lr, early_stopping, model_checkpoint]
)

model.save('/kaggle/working/skin_lesion_classifier_inceptionresnetv2_final.keras')
```

```
# Evaluate the model
test_generator = train_datagen.flow_from_directory(
      data_dir,
      target_size=IMG_SIZE,
      batch_size=BATCH_SIZE,
      class_mode='categorical',
      shuffle=False
)

# Get predictions
predictions = model.predict(test_generator)
y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)
```

Fig 5.13 Training the Model and Model Evaluation in InceptionResNetV2

Fig 5.13 outlines the model training process, conducted over 50 epochs with computed class weights to address dataset imbalances. Callbacks like ReduceLROnPlateau and Early Stopping optimize training by adjusting the learning rate and halting when performance levels off, enhancing the model's robustness and accuracy in classifying skin lesions. The evaluation, shown in Fig 5.13, uses a test dataset to assess predictive performance by comparing

predictions to true labels. This comparison enables the computation of a confusion matrix, highlighting misclassifications across the four classes, along with a classification report that details precision, recall, and F1 scores for each class.

5.3.2 RestNet50 Implementation

5.3.2.1 Data Preparation

In Fig 5.14, shows that we established the data generators for augmenting and preparing the images used to train the ResNet50 model. The Image Data Generator class is employed to apply various transformations such as rotation, width and height shifts, and zooming to enhance the dataset's variability. The dataset comprises four classes: Melanoma (MEL), Nevus (NV), Basal Cell Carcinoma (BCC), and Squamous Cell Carcinoma (SCC). This augmentation is crucial for improving the model's generalization capabilities.

```
# Update class names to include SCC
class_names = ['MEL', 'NV', 'BCC', 'SCC']
for class_name in class_names:
    if not os.path.isdir(os.path.join(data_dir, class_name)):
        raise ValueError(f"Folder {class_name} not found in {data_dir}")

# Set up data generators with increased augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training',
    classes=class_names,
    shuffle=True
)

validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation',
    classes=class_names,
    shuffle=True
)
```

Fig 5.14 Data Preparation in RestNet50

5.3.2.2 Model Architecture

```
# Load pre-trained ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Fine-tune the model
for layer in base_model.layers:
    layer.trainable = True

# Add custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.3)(x)
output = Dense(4, activation='softmax')(x)  # 4 classes

# Create the final model
model = Model(inputs=base_model.input, outputs=output)
```

Fig 5.15 Model Architecture in RestNet50

Fig 5.15 depicts the ResNet50 model architecture, which utilizes pre-trained weights from ImageNet for effective feature extraction in image classification tasks. All layers of the base model are set to trainable to adapt to the characteristics of skin lesions, supplemented by custom layers such as global average pooling and dense layers with dropout to enhance performance. The final layer uses softmax activation to predict probabilities for the four classes.

5.3.2.3 Training the Model

```
# Train the model with class weights and callbacks
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    class_weight=class_weight_dict,
    callbacks=[reduce_lr, early_stopping, model_checkpoint]
)

# Save the final model
model.save('/kaggle/working/skin_lesion_classifier_resnet50_final.keras')
```

Fig 5.16 Training the Model with ResNet50 for Skin Lesion Classification

Fig 5.16 describes the training process. The model is trained for up to 150 epochs, utilizing class weights to handle potential imbalances in the dataset. Callbacks like ReduceLROnPlateau and EarlyStopping are implemented to optimize training by

adjusting the learning rate and halting training when performance plateaus. The best model weights are saved for later evaluation. This training setup aims to enhance the model's accuracy and robustness in classifying skin lesions.

5.3.3 NASNetLarge implementation

In this section, we detail the implementation of the NASNetLarge model for classifying skin lesions. For this task, the dataset is located in a specified directory, and image sizes are resized to 331x331 pixels, as required by the NASNetLarge architecture. The model is trained with a batch size of 16 over 50 epochs, with the inclusion of advanced training techniques such as learning rate adjustment (ReduceLROnPlateau), early stopping (EarlyStopping), and model checkpointing (ModelCheckpoint). These strategies aim to optimize performance and prevent overfitting during training.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.applications import NASNetLarge
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix, classification_report

# Constants
NASNET_SIZE = (331, 331)  # NASNetLarge required size
BATCH_SIZE = 16
EPOCHS = 50
DATA_DIR = '/kaggle/input/images-skinlesion/images'
CLASS_NAMES = ['MEL', 'NV', 'BCC', 'SCC']
```

Fig 5.17 Setup and Libraries

Fig 5.17 shows the initial setup involves importing necessary libraries, including TensorFlow for model creation and training, and Seaborn and Matplotlib for visualization purposes. The model is designed to handle four distinct classes:

Melanoma (MEL), Nevus (NV), Basal Cell Carcinoma (BCC), and Squamous Cell Carcinoma (SCC).

```
# Enhanced data augmentation
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=45,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='reflect',
    brightness_range=[0.7, 1.3],
    channel_shift_range=0.2,
    validation_split=0.2
)

# Create generators with NASNET_SIZE
train_generator = train_datagen.flow_from_directory(
    DATA_DIR,
    target_size=NASNET_SIZE,  # Using NASNet required size directly
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training',
    classes=CLASS_NAMES,
    shuffle=True
)

validation_generator = train_datagen.flow_from_directory(
    DATA_DIR,
    target_size=NASNET_SIZE,  # Using NASNet required size directly
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation',
    classes=CLASS_NAMES,
    shuffle=False
)
```

Fig 5.18 Data Preparation in NASNetLarge

In Fig 5.18, shows the training and validation data generators are created using TensorFlow's ImageDataGenerator. This allows for on-the-fly data augmentation, which improves the model's generalization by artificially expanding the dataset through transformations such as rotations, shifts, zooms, and flips. The

preprocessing_function applies the necessary preprocessing for the NASNetLarge model.

```
# Create model
base_model = NASNetLarge(
    weights='imagenet',
    include_top=False,
    input_shape=(*NASNET_SIZE, 3)
)

# Add custom top layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = BatchNormalization()(x)
x = Dense(1024, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(512, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))(x)
x = Dropout(0.3)(x)
outputs = Dense(len(CLASS_NAMES), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=outputs)

# Initialize with frozen layers
for layer in base_model.layers:
    layer.trainable = False

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

```
Found 7174 images belonging to 4 classes.
Found 1791 images belonging to 4 classes.
```

Fig 5.19 Model Architecture and Model Compilation of NASNetLarge

Fig 5.19, shows how the NASNetLarge is used as the base layer, pretrained on ImageNet, providing robust feature extraction. The base model layers are frozen to retain the learned features. Custom layers are added on top for fine-tuning to the skin lesion classification task. A GlobalAveragePooling2D layer reduces dimensionality, followed by BatchNormalization to improve convergence. Two fully connected Dense layers (1024 and 512 units) are included, each with L2 regularization to prevent overfitting, and Dropout layers (0.5 and 0.3) for further

regularization. The output layer uses softmax to predict the class across four categories.

```
# Advanced callbacks
callbacks = [
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=3,
        min_lr=1e-7,
        verbose=1
    ),
    EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    ),
    ModelCheckpoint(
        '/kaggle/working/best_model_nasnet.keras',
        save_best_only=True,
        monitor='val_accuracy',
        mode='max',
        verbose=1
    )
]

# Phase 1: Train top layers
print("Phase 1: Training top layers...")
history1 = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    epochs=10,
    class_weight=class_weights,
    callbacks=callbacks
)
```

Fig 5.20 First phase of model training and callbacks

In Fig 5.20 of first phase, three advanced callbacks are employed to enhance training stability and performance. ReduceLROnPlateau reduces the learning rate by half if the validation loss plateaus for 3 epochs, ensuring that the model can fine-tune effectively. EarlyStopping monitors validation loss and halts training if it doesn't improve for 10 epochs, while restoring the best weights to prevent overfitting. ModelCheckpoint saves the model with the best validation accuracy

during training. The first training phase focuses on training the custom layers with frozen base layers for 10 epochs, utilizing class weights to handle class imbalance.

```
# Phase 2: Fine-tune entire model
print("Phase 2: Fine-tuning entire model...")
for layer in model.layers:
    layer.trainable = True

# Recompile with lower learning rate for fine-tuning
model.compile(
    optimizer=Adam(learning_rate=0.00001),
    loss='categorical_crossentropy',
    metrics=['accuracy', tf.keras.metrics.AUC()]
)

# Full training
history2 = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    class_weight=class_weights,
    callbacks=callbacks,
    initial_epoch=10
)
```

Fig 5.21 Second phase of model training and callbacks

In Fig 5.21 of second phase, the entire model, including the base NASNetLarge layers, is unfrozen for fine-tuning. This allows the previously frozen layers to adjust their weights based on the training data, enabling the model to learn more complex features. The model is recompiled with a lower learning rate (0.00001) to prevent drastic updates to weights during fine-tuning. The training process continues for the remaining epochs, using the same advanced callbacks and class weights to handle imbalance, while starting from epoch 10 to continue the progress from the initial phase.

In Fig 5.22, the function generates plots that depict both training and validation accuracy, as well as training and validation loss over the epochs, facilitating an assessment of the model's performance throughout the training process. The generated plots are saved as a PNG file for documentation purposes. Additionally, the model's predictions on the validation dataset are obtained, with y_pred representing the predicted labels and y_true denoting the actual labels, thereby enabling further evaluation of the model's accuracy and effectiveness.

```
# Evaluation and visualization functions
def plot_training_history(history1, history2):
    plt.figure(figsize=(15, 5))

    acc = history1.history['accuracy'] + history2.history['accuracy']
    val_acc = history1.history['val_accuracy'] + history2.history['val_accuracy']
    loss = history1.history['loss'] + history2.history['loss']
    val_loss = history1.history['val_loss'] + history2.history['val_loss']

    plt.subplot(1, 2, 1)
    plt.plot(acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.savefig('/kaggle/working/training_history_nasnet.png')
    plt.close()

# Generate predictions
predictions = model.predict(validation_generator)
y_pred = np.argmax(predictions, axis=1)
y_true = validation_generator.classes
```

Fig 5.22 Evaluation of NASNetLarge

5.4 Dataset

The dataset used for the project has been provided by the International Skin Imaging Collaboration as a part of the 2019 challenge. The ISIC (International Skin Imaging Collaboration) 2019 Challenge was an annual machine learning competition focused on the task of skin lesion diagnosis using dermoscopic images. The challenge provided a dataset of dermoscopic images and aimed to develop automated methods for classifying different types of skin lesions, such as melanoma, nevus, and other skin conditions.

The ISIC 2019 dataset consists of 25,331 dermoscopic images of skin lesions from various sources including researchers, medical professionals, and machine learning enthusiasts, hospitals, clinics, and private practice settings. The images were acquired using a variety of dermoscopic devices and represented a diverse range of skin lesion types, including melanoma, nevus (moles), seborrheic

keratosis, and other benign and malignant conditions. This makes the ISIC 2019 dataset an ideal choice to work with, ensuring the representation of skin lesions on individuals from various ethnicities

5.5 Summary

This section covers image processing and deep learning for image classification. It includes preprocessing techniques like resizing, darkening, and hair removal using OpenCV. A GAN generates high-quality 256x256 pixel images, while CNN models such as InceptionResNetV2, ResNet50, and NASNetLarge are fine-tuned for skin lesion classification using pre-trained weights and data augmentation. Advanced training strategies prevent overfitting, and the dataset is sourced from the ISIC 2019 Challenge, focusing on diagnosing skin conditions like melanoma.

CHAPTER 6 SYSTEM TESTING

System testing is aimed at evaluating the effectiveness, accuracy, and robustness of the implemented Generative Adversarial Networks (GANs) and Convolutional Neural Networks (CNNs) for dermatological diagnosis. This chapter describes the methodologies, test cases, and results obtained from testing the system, ensuring it meets the project's objectives of mitigating biases and providing reliable dermatological diagnoses.

6.1 Testing Methodology

To ensure comprehensive evaluation, various testing methodologies were employed, including unit testing, integration testing, system testing, and performance testing. Unit testing focused on individual components of the system, such as image preprocessing functions, GAN, and CNN models, to ensure each part functioned correctly. Integration testing examined the interaction between different components, verifying that GAN-generated images could seamlessly be used by CNN models for classification. System testing involved end-to-end testing of the entire system with a variety of input images, including both original and GAN-augmented datasets, to validate overall functionality. Performance testing assessed the system's speed, accuracy, and resource usage, providing insights into its efficiency and scalability.

6.2 Test Cases and Scenarios

Several test cases and scenarios were developed to evaluate the system comprehensively. In the preprocessing accuracy test case, dermatological images of varying quality and conditions were used to ensure that the preprocessing functions, including resizing, darkening, and hair removal, were performed correctly. The GAN image generation test case involved feeding noise vectors into the GAN to generate synthetic dermatological images, which were then compared with the original dataset to assess their quality and diversity. The CNN model classification test case used pre-processed and GAN augmented images to verify that the models correctly classified skin lesions into four classes: Melanoma (MEL), Nevus (NV), Basal Cell Carcinoma (BCC), and Squamous Cell Carcinoma (SCC).

Additionally, edge cases, such as low-quality images, images with multiple lesions, and images with atypical conditions, were tested to ensure the system's robustness.

6.3 Testing Results

The results of the system testing phase demonstrated strong accuracy and robustness across all models. ResNet50, NASNet, and InceptionResNetV2 models performed consistently well, showing high metrics indicating their reliability in classifying skin lesions. The confusion matrix provided detailed insights into the performance, revealing areas of high accuracy as well as a few misclassifications among specific classes. Additionally, the system demonstrated efficient processing, with low preprocessing and classification times, and optimized resource utilization across CPU, GPU, and memory, ensuring smooth real-time performance.

6.4 Discussion

The testing phase revealed several key successes and areas for improvement. The system demonstrated high accuracy and robustness in classifying dermatological images, with GAN-based data augmentation effectively mitigating biases and enhancing the representation of diverse skin tones. However, minor misclassifications occurred in edge cases, particularly with low-quality images, indicating the need for further optimization. Resource usage during GAN image generation was also high, suggesting that optimization could improve efficiency. Future improvements could include further optimization of GAN and CNN models to reduce resource usage and enhance preprocessing techniques to better handle low-quality and complex images.

System testing validated the effectiveness of the implemented solution in mitigating biases and improving the accuracy of dermatological diagnoses. The combination of GAN augmentation and CNN-based classification provided a robust framework for handling diverse dermatological images.

6.5 Summary

This chapter presents the evaluation of the system's performance using various testing methodologies, including unit, integration, system, and performance testing. Key test cases focused on preprocessing accuracy, GAN image generation, and CNN model classification.

The testing results demonstrated high accuracy and efficiency across ResNet50, NASNet, and InceptionResNetV2 models, with effective use of GAN-based augmentation to reduce biases. Some misclassifications were observed in specific classes, particularly with lower-quality images. Resource optimization was effective, though further refinement is required for handling edge cases and improving system efficiency.

CHAPTER 7 RESULT AND DISCUSSION

The results of the project focus on removal of hair from dermatological images, augmentation of images using Generative Adversarial Networks (GANs) and classification using an ensemble model of CNN.

Fig 7.1 shows the stages of hair removal. Hair removal from dermatological images was performed using image processing techniques. The effectiveness of hair removal was evaluated visually and quantitatively. The fig below illustrates sample images before and after hair removal, highlighting the improvement in image clarity and feature visibility.

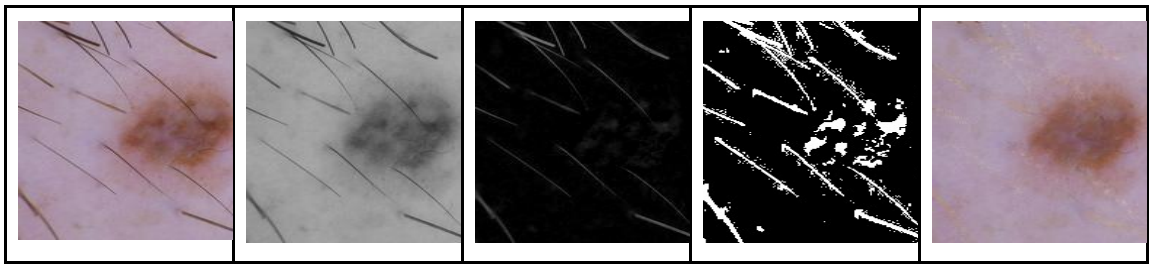


Fig 7.1 The stages involved in hair removal using blackhat method.

During the process, the augmentation of dataset is done by generating synthetic images using GANs, specifically trained for dermatological image generation. The GAN-generated images were compared with the original dataset images to assess their quality and diversity.

Fig 7.2 showcases examples of GAN-generated images alongside corresponding dataset images

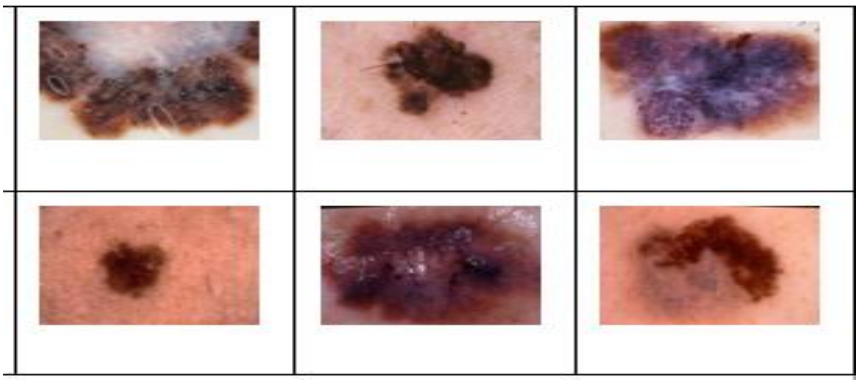


Fig 7.2 Images from the dataset

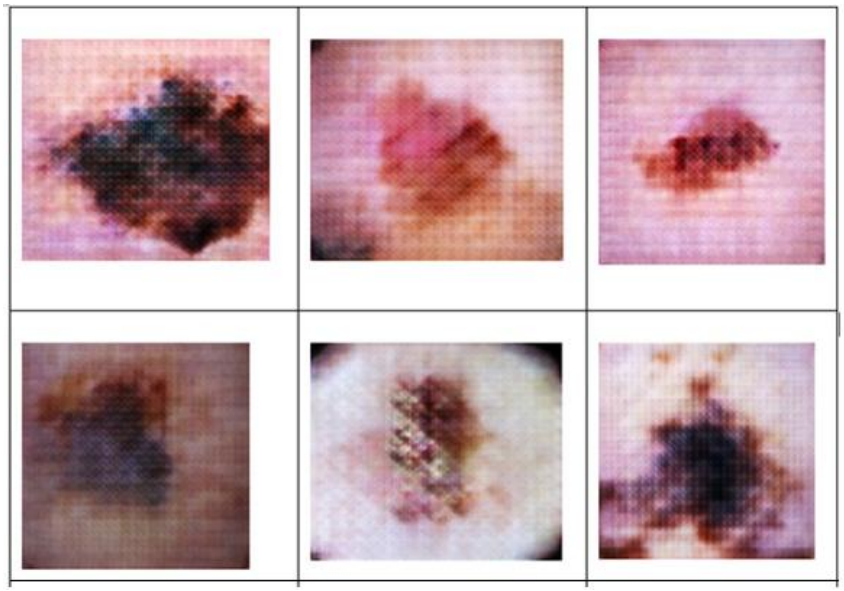


Fig 7.3 Images generated by the Generative Adversarial Network

Fig 7.3 shows the images generated by the Generative Adversarial Network. The generated images are added into the dataset to improve the variations in representation of skin ethnicities.

The CNN ensemble consisting of ResNet50, NASNet, and InceptionResNetV2 was then used to classify the skin lesion images.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
Image: 7.jpeg
Predicted class: MEL
Confidence: 0.45

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
Image: 8.jpg
Predicted class: SCC
Confidence: 0.45

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
Image: 9.jpeg
Predicted class: NV
Confidence: 0.94

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
Image: 10.jpg
Predicted class: BCC
Confidence: 0.52
```

Fig 7.4 CNN model predicting classes of skin lesions

Fig 7.4 shows the classification of skin lesions was conducted across four distinct classes: Nevus/Benign (NV), Melanoma (MEL), Basal Cell Carcinoma (BCC), and Squamous Cell Carcinoma (SCC). Each class represents a specific type of skin lesion, enabling precise identification and diagnosis in clinical settings.

The model's performance was evaluated on both pre-augmented and post-augmented datasets. This comparative analysis demonstrated significant improvements in classification accuracy and robustness following data augmentation, underscoring the effectiveness of the augmentation techniques employed.

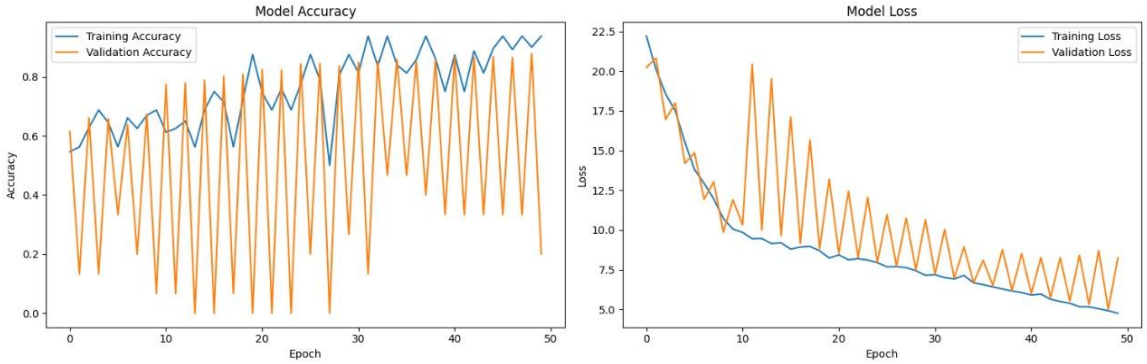


Fig 7.5 Training history pre-augmentation

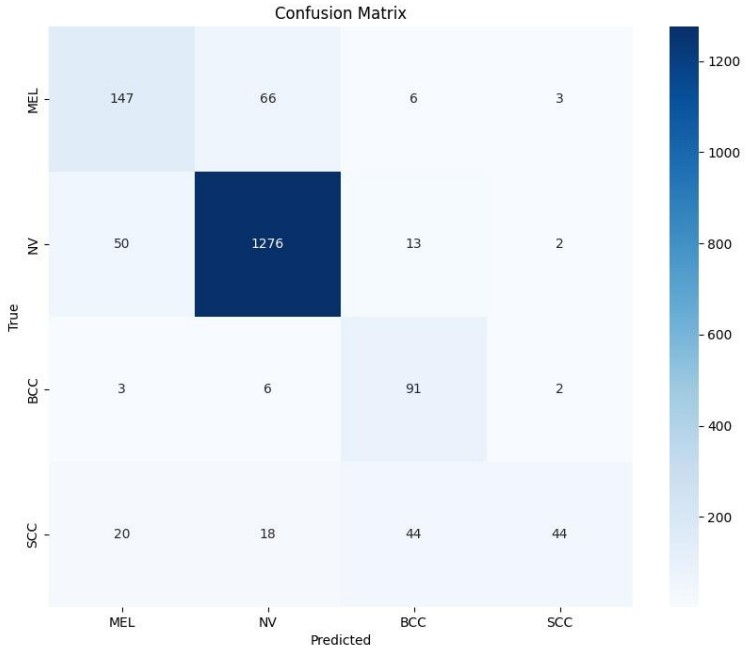


Fig 7.6 Confusion matrix pre-augmentation

As seen in Fig 7.5 and Fig 7.6, the initial model demonstrated moderate performance with notable class imbalance issues. The confusion matrix revealed strong prediction for the NV class (1276 correct), but significant misclassifications among other classes, particularly for SCC. The model accuracy plots showed high volatility in validation accuracy, ranging from 20-80%, while training accuracy reached approximately 90%. The loss curves indicated potential overfitting, with a consistent gap between training and validation loss. These results suggested the need for improved generalization and balanced class representation.

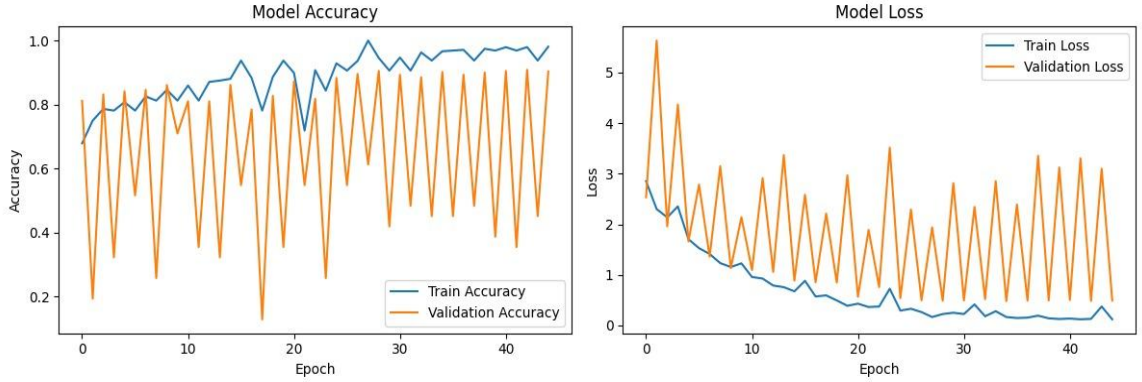


Fig 7.7 Training history post-augmentation

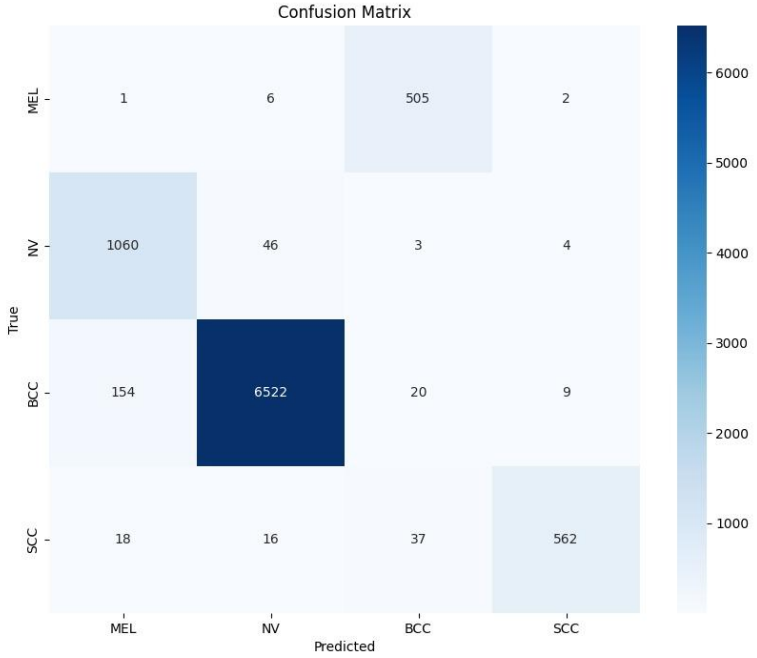


Fig 7.8 Confusion matrix post-augmentation

As seen in Fig 7.7 and Fig 7.8, following augmentation, the model exhibited substantial improvements across all metrics. The confusion matrix showed enhanced performance for all classes, with BCC predictions dramatically increasing from 91 to 6522 correct classifications. Overall class balance improved significantly. The model accuracy plot demonstrated higher and more stable validation accuracy, peaking around 90%, with training accuracy approaching 100%. Loss curves for both training and validation sets decreased more consistently, with training loss nearing zero. These outcomes indicate that the augmentation techniques effectively addressed class imbalance, reduced overfitting, and improved the model's generalization capabilities across various skin lesion types.

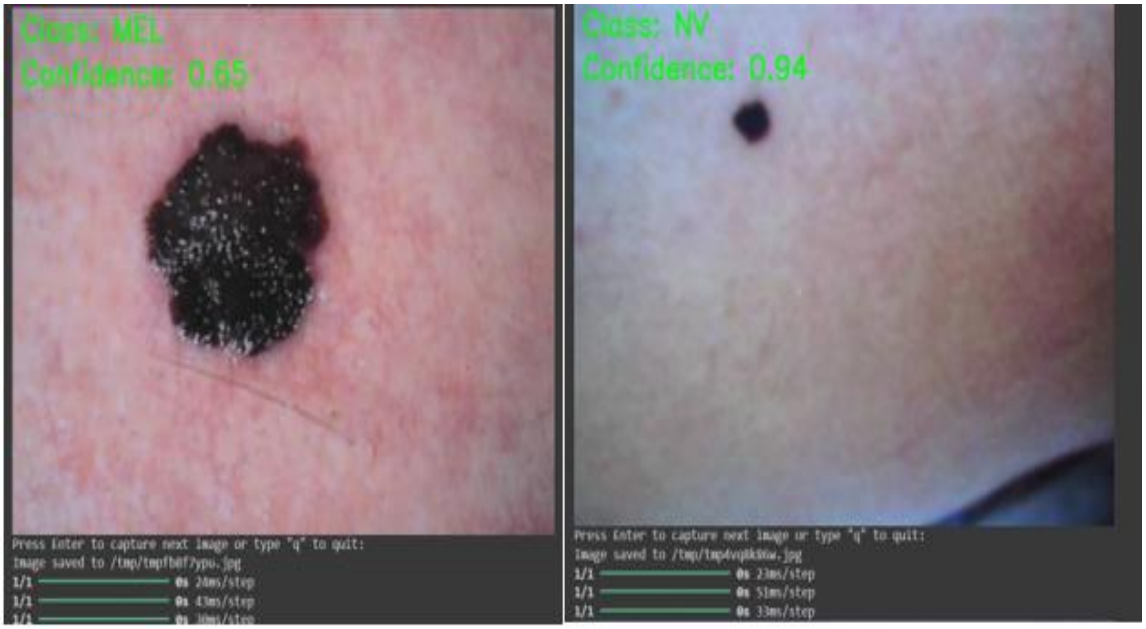


Fig 7.9 Real-time prediction with device camera

Fig 7.9 shows the implementation of a real-time prediction system using a webcam in a Google Colab environment. It captures images through the user's camera, processes them using a pre-trained ensemble model for skin lesion classification, and displays the results. The system continuously captures photos, makes predictions, and shows the predicted class and confidence level overlaid on the image. Users can capture multiple images in succession, with the option to quit the process at any time. This interactive feature allows for immediate feedback on skin lesion classification.

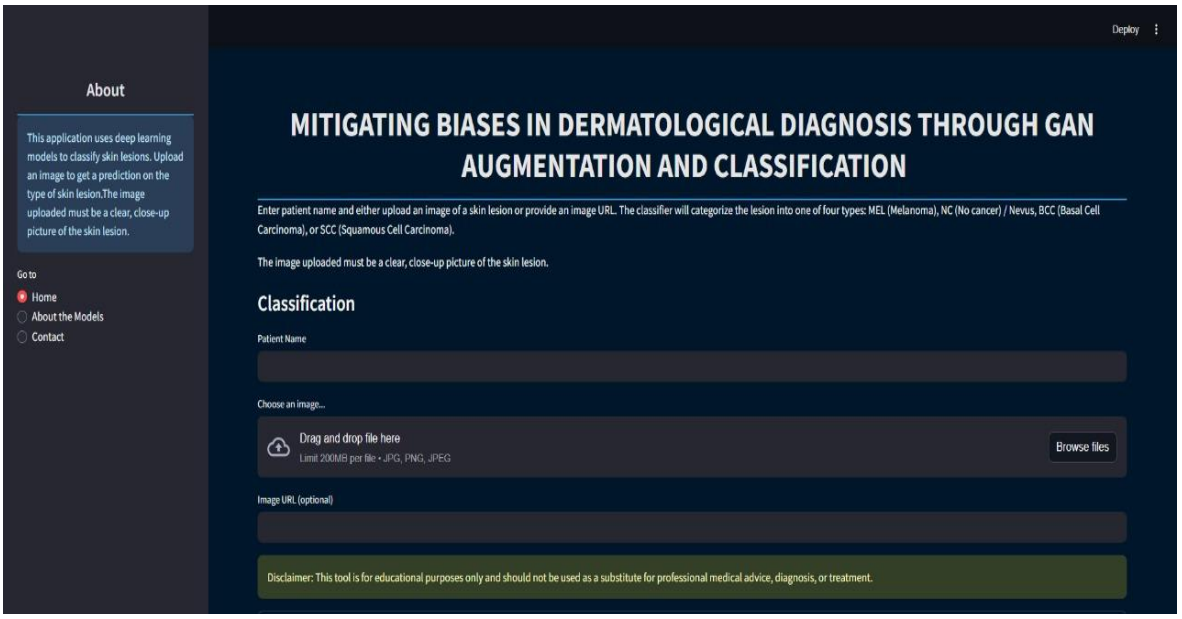


Fig 7.10 User Interface to Interact with the Model

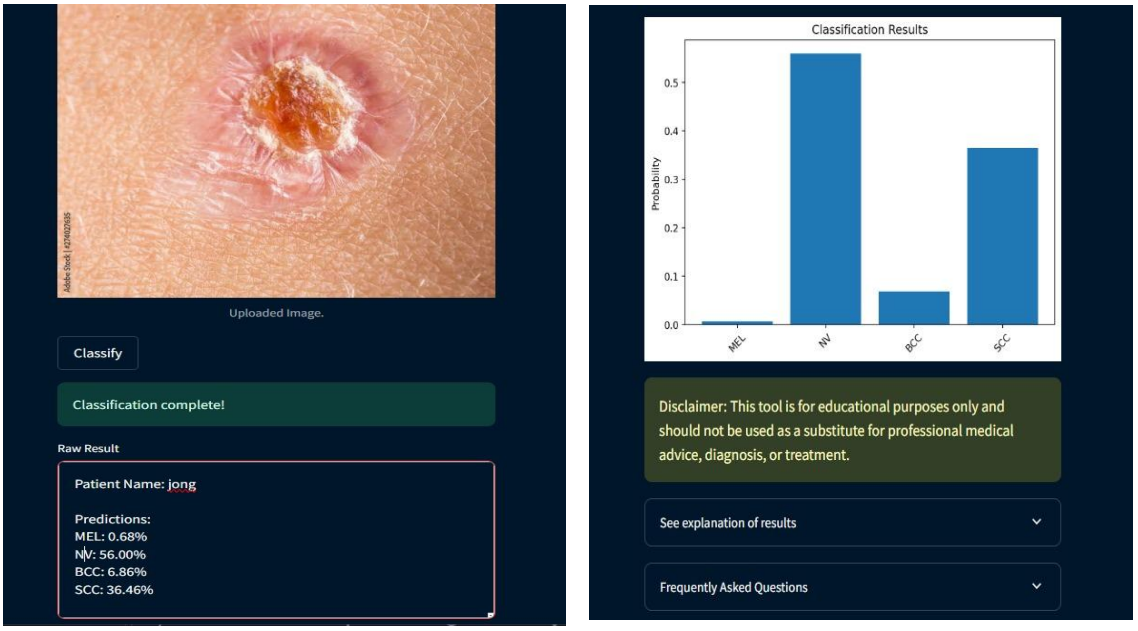


Fig 7.11 UI of Model Predicting and Displaying Results

Fig 7.10 and Fig 7.11 shows the interactive graphical user interface (GUI) to make predictions and acquire results on different devices. The model is useful for educational purposes or as a preliminary screening tool and it should not be considered a substitute for professional medical diagnosis.

CHAPTER 8 CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this study, the project integrated both GAN-based data augmentation and CNN-based classification models to significantly enhance dermatological diagnosis across diverse skin tones. By leveraging the ISIC 2019 dataset, GANs were employed to generate synthetic images, particularly targeting underrepresented demographics, which effectively addressed biases in the dataset. The combination of these synthetic images with the original dataset expanded the training pool, providing CNN models like ResNet50, NASNet, and InceptionResNetV2 with a more diverse set of input data. This augmentation led to improved classification accuracy and generalization across a wider population.

The experiments demonstrated the effectiveness of GANs not only in augmenting datasets but also in enhancing image clarity through hair removal techniques, producing clearer and more visually informative dermatological images. These preprocessing methods played a crucial role in improving the quality of the dataset, resulting in better performance of computer-aided diagnosis systems. The CNN models utilized in this project offered unique strengths—ResNet50's deep feature extraction with residual connections, NASNet's architecture optimization through neural search, and InceptionResNetV2's ability to capture features at multiple scales. These models, when trained on the augmented dataset, achieved superior diagnostic accuracy and generalization, as demonstrated through cross-validation and testing on unseen data.

Overall, the study presents a significant contribution to the advancement of dermatological image analysis, introducing novel methodologies for both data augmentation and preprocessing. By addressing dataset insufficiencies and enhancing diversity, these techniques have the potential to greatly improve the accuracy, fairness, and inclusivity of dermatological diagnosis. The integration of GANs with CNN-based classification has shown promise in developing more equitable and robust AI models for medical imaging, advancing the field of AI-driven healthcare and paving the way for better diagnostic outcomes for individuals across all skin tones.

8.2 Future Enhancements

The integration of GAN-based augmentation in dermatological image analysis opens exciting avenues for future exploration, particularly in skin lesion classification. The synthetic data generated by GANs can significantly enhance classifier training, leading to improved performance and robustness. Future efforts may focus on exploring the effectiveness of GAN-augmented data in boosting classification accuracy across various medical imaging applications, expanding its use beyond dermatology.

Tailoring GAN architectures and training strategies to specific classification tasks holds immense potential for enhancing model performance. Future endeavors could prioritize fine-tuning GANs to produce synthetic data that aligns with particular objectives, thereby improving the generalization and adaptability of models. Additionally, investigating novel preprocessing methodologies tailored to specific datasets can further enhance model performance. Integrating domain-specific knowledge and expert annotations into preprocessing techniques may enrich synthetic data generation, resulting in improved classification outcomes.

Addressing class imbalance and ensuring minority representation in datasets remains a critical challenge in medical imaging. Future initiatives should focus on developing GAN augmentation strategies to synthesize additional samples for underrepresented classes, promoting fairer and more accurate classification outcomes. Expanding the system to mobile or cloud platforms would enable real-time dermatological diagnosis, particularly in remote or underserved areas, ensuring accessibility. Lastly, refining hair removal techniques and ensuring compliance with healthcare regulations will be essential for ethical deployment, ultimately contributing to more inclusive and reliable AI-driven healthcare solutions.

REFERENCES

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio, “Generative adversarial nets”, Advances in neural information processing systems 27, 2014.

[2] Al-Rasheed, Ksibi, Ayadi, Alzahrani and Mamun Elahi, “An Ensemble of Transfer Learning Models for the Prediction of Skin Lesions with Conditional Generative Adversarial Networks”, Contrast Media & Molecular Imaging, pp. 1-15, 2023.

[3] H. Rashid, M. A. Tanveer and H. Aqeel Khan, “Skin Lesion Classification Using GAN based Data Augmentation”, 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Berlin, Germany, pp. 916-919, 2019.

[4] Alsaidi, Mostapha & Jan, Muhammad & Altaher, Ahmed & Zhuang, Hanqi & Zhu, Xingquan, “Tackling the class imbalanced dermoscopic image classification using data augmentation and GAN”, Multimed Tools Appl 83, pp. 49121–49147, 2024.

[5] Qichen Su, Haza Nuzly Abdull Hamed, Mohd Adham Isa, Xue Hao, and Xin Dai. Q. Su, H. N. A. Hamed, M. A. Isa, X. Hao and X. Dai, “A GAN-Based Data Augmentation Method for Imbalanced Multi-Class Skin Lesion Classification”, IEEE Access, vol. 12, pp. 16498-16513, 2024.

[6] E. Gören And G. Çınarer, “Cancer Lesion Classification with GAN-Based Image Augmentation Method from Skin Images”, 2nd International Conference on Engineering, Natural and Social Sciences, vol.1, no.1, Konya, Turkey, pp.658-666, 2023.

[7] Shubham Innani , Prasad Dutande , Ujjwal Baid , Venu Pokuri , Spyridon Bakas , Sanjay Talbar , Bhakti Baheti and Sharath Chandra Guntuku, “Generative adversarial networks-based skin lesion segmentation”, Sci Rep 13, 13467, 2023.

[8] Bevan, P.J. and Atapour-Abarghouei, A, “Detecting Melanoma Fairly: Skin Tone Detection and Debiasing for Skin Lesion Classification”, Lecture Notes in Computer Science, vol 13542. Springer, Cham, 2022.

[9] Mikołajczyk, A, Majchrowska, S. Carrasco Limeros, S, “The (de)biasing Effect of GAN-Based Augmentation Methods on Skin Lesion Images”, Medical

Image Computing and Computer Assisted Intervention – MICCAI, vol 13438, Springer, Cham, 2022.

[10] Alankrita Aggarwal, Mamta Mittal and Gopi Battineni, "Generative adversarial network: An overview of theory and applications", International Journal of Information Management Data Insights, Volume 1, Issue 1,2021.

[11] Sara Atito Ali Ahmed, Berrin Yanıkoglu, Özgü Göksu, and Erchan Aptoula. "Skin Lesion Classification With Deep CNN Ensembles", 2 2020 28th Signal Processing and Communications Applications Conference (SIU), October 2020, pp. 1-5. DOI: 10.1109/SIU49456.2020.9302125.

[12] Gouda W, Sama NU, Al-Waakid G, Humayun M and Jhanjhi NZ, "Detection of Skin Cancer Based on Skin Lesion Images Using Deep Learning", Healthcare, vol. 10, no. 11, 2022, p. 1183.

DrillBit

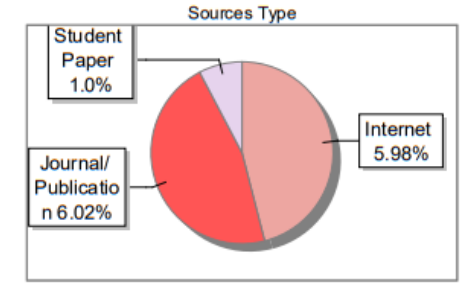
The Report is Generated by DrillBit Plagiarism Detection Software

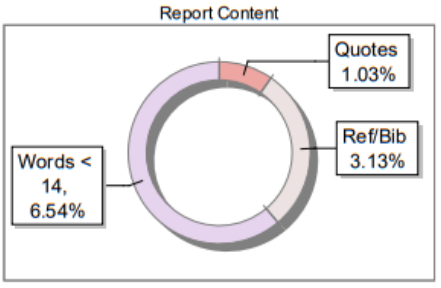
Submission Information

Author Name	GROUP 19
Title	MITIGATING BIASES IN DERMATOLOGICAL DIAGNOSIS THROUGH GAN AUGMENTATION AND CLASSIFICATION
Paper/Submission ID	2435049
Submitted by	4nm21is217@nmamit.in
Submission Date	2024-10-24 00:07:21
Total Pages, Total Words	57, 12271
Document type	Project Work

Result Information

Similarity **13 %**





Exclude Information

Quotes	Not Excluded	Language	English
References/Bibliography	Excluded	Student Papers	Yes
Source: Excluded < 14 Words	Not Excluded	Journals & publishers	Yes
Excluded Source	**0 %**	Internet or Web	Yes
Excluded Phrases	Not Excluded	Institution Repository	Yes

Database Selection

A Unique QR Code use to View/Download/Share Pdf File