DEEP NEURAL NETWORKS IN THE FIGHT AGAINST MELANOMA

A PROJECT REPORT

Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPURAMU

In partial fulfillment of the requirements for the award of the degree of
Master of Technology
In

Computer Science and Engineering By

ANITHA KARRE

(23G31D5808)

Under the guidance of

T. Abdul Raheem, M. Tech.

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING ST.

JOHNS COLLEGE OF ENFINERRING & TECHNOLOGY

(Autonomous)

(Affiliated to JNTUA, Anantapuramu and approved by AICTE, New Delhi) Yerrakota,

Yemmiganur, Kurnool (Dist)-518 360, A.P.

2023 – 2025

ST.JOHNS COLLEGE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

(Affiliated to JNTUA, Anantapuramu and approved by AICTE, New Delhi)
Yerrakota, Yemmiganur, Kurnool (Dist)-518 360, AP.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the thesis entitled **"Deep Neural Networks in the fight against Melanoma"** is bonafide work of **Karre Anitha** bearing Admission no**: 23G31D5808** submitted to the Department of Computer Science & Engineering, in partial fulfillment of the requirements for the award of degree of **MASTER OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING f**rom **Jawaharlal Nehru Technological University, Anantapuramu.**

Signature of the Supervisor.	Signature of the Head of the Dept.
T. ABDUL RAHEEM M. Tech.	**Dr. P. VEERESH, Ph. D.**
Assistant Professor	**Professor, H.O.D**
Department of CSE	**Department of CSE**
St. Johns College of Engineering and Technology	**St. Johns College of Engineering and Technology**

ST. JOHNS COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous)

(Affiliated to JNTUA, Anantapuramu and approved by AICTE, New Delhi)
Yerrakota, Yemmiganur, Kurnool (Dist)-518 360, AP.

DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING DECLARATION

I hereby declare that the project entitled **"DEEP NEURAL NETWORKS IN THE FIGHT AGAINST MELANOMA"** submitted by me to the Department of Computer Science and Engineering, **St. Johns College of Engineering &Technology**, **Yerrakota, Yemmiganur, Kurnool,** in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Science and Engineering** is a record of bonafide work carried out by me under the supervision of Assistant Professor **T. ABDUL RAHEEM,** I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or any other institute or university.

Signature

KARRE. ANITHA

(23G31D5808)

ST. JOHNS COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous)

(Affiliated to JNTUA, Anantapuramu and approved by AICTE, New Delhi)
Yerrakota, Yemmiganur, Kurnool (Dist)-518 360, AP.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

The project report entitled **"DEEP NEURAL NETWORKS IN THE FIGHT AGAINST MELANOMA"** is prepared and submitted by **KARRE. ANITHA (23G31D5808).** It has been found satisfactory in terms of scope, quality and presentation as partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Science and Engineering** in St. Johns College of Engineering & Technology, Yerrakota, Yemmiganur, Kurnool, A.P.

Guide **Head of the Department**

Internal Examiner **External Examiner**

ACKNOWLEDGEMENTS

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that I have now the opportunity to express my guidance for all of them.

The first and foremost, **T. ABDUL RAHEEM,** Assistant Professor of Computer Science and Engineering Department, who has extended his support for the success of this project. His wide knowledge and logical way of thinking have made a deep impression on me. His understanding, encouragement and personal guidance have provided the basis for this thesis. His source of inspiration for innovative ideas and his kind support is well to all his students and colleagues.

I wish to thank **Dr. P. VEERESH,** Head of Computer Science and Engineering Department. His wide support, knowledge and enthusiastic encouragement have impressed me to better involvement into my project thesis and technical design also his ethical morals helped me to develop my personal and technical skills to deploy my project in success.

I wish to thank **Dr. K. SUDHAKAR**, Principal of St. Johns College of engineering and Technology who has extended his support for the success of this project.

I express my sincere thanks to the project committee members, faculty and staff of Computer Science and Engineering Department, St. Johns College of engineering and Technology, for their valuable guidance and technical support.

Last but far from least, I also thank my family members and my friends for their moral support and constant encouragement, I am very much thankful to one and all who helped me for the successful completion of the project

With gratitude

KARRE. ANITHA (20G31D5808)

INDEsX

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The increasing prevalence of skin cancer necessitates the development of effective diagnostic tools to aid in early detection and treatment. This project aims to leverage deep learning techniques, specifically a custom Convolutional Neural Network (CNN), to classify various skin diseases using a dataset from the International Skin Imaging Collaboration (ISIC). The dataset comprises 2,357 images categorized into nine distinct classes, convering both malignant and benign skin disorders, including melanoma, basal cell carcinoma, and actinic keratosis. The primary objective of this project is to build a robust multiclass classification model that accurately identifies and classifies these skin diseases. The project will follow a structured pipeline, beginning with data reading and understanding, followed by dataset creation and visualization.

A CNN model will be constructed and trained on the dataset, with careful attention to normalization, optimization, and loss functions. To enhance model performance and mitigate issues of overfitting and underfitting, data augmentation techniques will be employed. Additionally, class distribution will be analyzed to address any imbalances within the dataset, making certain the model is trained on a diverse set of all categories of classes. By the end of this project, we aim to develop a reliable classification model that can assist healthcare professionals in diagnosing skin diseases, ultimately contributing to improved patient outcomes and reducing the potential risks associated with the incorrect detection, leading to false alarms or delayed diagnosis and misclassification of skin cancer.

CHAPTER – 1

INTRODUCTION

1.1 OVERVIEW

The skin serves as a vital barrier, regulating body temperature and protecting against environmental factors, including sunlight. However, it is susceptible to various diseases, including malignant tumors like melanoma, squamous cell carcinoma, and basal cell carcinoma, which collectively account for over 3.5 million diagnoses annually. Melanoma, one of the most aggressive skin cancers, arises from melanocytes and is often characterized by block or brown growths, though it can also appear in other colors. Studies show that 86% of melanomas are associated with ultraviolet (UV) radiation exposure, with the risk increasing twofold after five sunburns. Early detection is vital, offering a cure rate of up to 90%. Conventional diagnosis typically depends on biopsy, which can be both slow and inaccurate.

Noninvasive techniques, such as dermoscopic imaging, have become essential for dermatologists, providing high-resolution images that reveal deeper skin layers. Recent advancements in computer vision and digital image processing, particularly through Artificial Intelligence (AI) and Deep Learning, have significantly improved the detection of various skin diseases, including actinic keratosis, basal cell carcinoma, dermatofibroma, melanoma, nevus, pigmented benign keratosis, seborrheic keratosis, squamous cell carcinoma, and vascular lesions.

Various methodologies, such as Convolutional Neural Networks (CNNs), decision trees, and random forests, have demonstrated high accuracy in classifying these conditions. The integration of these advanced methodologies into clinical practice can aid dermatologists in making informed decisions, ultimately improving patient outcomes. Additionally, innovative screening devices have been developed to efficiently facilitate the timely detection of skin diseases and to prevent complications, showcasing the immense potential of AI in dermatology.

The use of AI-driven models, particularly deep learning techniques, allows for real-time and accurate diagnosis, reducing the dependency on traditional biopsy methods. These models analyze a vast dataset of dermoscopic images to identify patterns and anomalies indicative of various skin conditions. AI systems enhance diagnostic accuracy by minimizing human errors and improving consistency in detecting skin diseases. Additionally, they can be incorporated into mobile applications, making them accessible to a wider population, especially in remote areas where dermatological expertise may be limited.

Moreover, deep learning models continuously improve over time through training with new datasets, ensuring that they adapt to emerging variations in skin diseases. AI-based tools also assist in prioritizing patients based on the severity of their conditions, allowing healthcare professionals to allocate resources more efficiently. These advancements highlight the transformative potential of AI in dermatology, paving the way for faster, more reliable, and cost-effective diagnostic solutions that can save lives through early intervention.

Furthermore, the adoption of AI-powered tools in dermatology is paving the way for telemedicine solutions, allowing remote diagnosis and consultation for patients who may not have immediate access to dermatologists. Mobile applications and cloud-based AI models are being developed to analyze skin lesions using smartphone cameras, providing preliminary assessments and guiding users toward professional medical advice. These advancements not only improve accessibility to dermatological care but also empower individuals to monitor their skin health proactively. As AI continues to evolve, its integration with dermatology holds the promise of making skin disease detection more efficient, accurate, and widely accessible on a global scale.

CHAPTER – 2

LITERATURE REVIEW

2.1 Content extraction studies using convolutional neural network

The growing prevalence of skin diseases, particularly melanoma, demands more efficient diagnostic methods. Traditional techniques like biopsy are slow and imprecise, highlighting the need for automated solutions. Method of Analysis: This study explores the use of Convolutional Neural Networks (CNNs) to classify dermatological images, including conditions such as melanoma, basal cell carcinoma, and actinic keratosis. CNNs extract key visual features from images, enabling accurate classification. Findings: The CNN model demonstrated high accuracy in identifying melanoma by detecting color variation, irregular borders, and other characteristic patterns. Novelty/Improvement: By automating the process, the CNN reduces human error, speeds up diagnosis, and enhances decision-making for dermatologists, ultimately improving patient outcomes.

2.2 Forecasting through Advanced Machine Learning Strategies

"As students progress through their academic journey and pursue courses of interest, it is crucial for them to evaluate their abilities and identify their passions. This helps them understand which career paths align with their skills and interests, ultimately guiding them toward a suitable profession. Such self-assessment fosters improved performance and fuels motivation, steering them toward their desired career goals. Additionally, recruiters, after evaluating candidates across various criteria, can user career recommendation system to determine the most appropriate job roles based on candidates performance and evaluations. This paper focuses primarily on predicting career paths for candidates in the Computer science field."

S. Harris, D. Kelly, V. Yang (2022), " DeepDerm"

Abstract: DeepDerm is an advanced deep learning model tailored specifically designed for automatic skin disease detection using Convolutional Neural Networks. With increasing prevalence of skin cancers and dermatological conditions, there is a growing demand for AI-powered diagnostic tools that can aid dermatologists in early detection, classification of skin diseases. DeepDerm addresses this challenge by employing a multi-class system that can accurately detect various skin disorders, including melanoma, basel cell carcinoma squamous cell carcinoma , eczema , warts and acne . The model is built on a deep CNN architecture inspired by VGG16, incorporating data augmentation techniques to increase generalization and dropout layers to mitigate overfitting. The dataset consists of over 50,000 high resolution images, ensuring a diverse representation of skin conditions. The Training pipeline includes adaptive learning rates and batch normalization, optimizing the Model for high sensitivity and specificity. While DeepDerm achieves high accuracy, it has (accuracy=91%, F1-Score=0.89, AUC=0.92) notable limitations. The model segmentation

Approach is very basic, less effective for detecting small lesions, training dataset is imbala nced, leading potential misclassifications in underrepresented conditions. Using class rebala ncing techniques to reduce bias and improve recognition across different skin types and conditions. Enhancing feature extraction, making the model more robust to real-world dermatological cases, where images might have poor lighting, occlusions, or noise.

J. Patel, M. Gonzalez (2021) “MobileNet-Skin”

Abstract: MobileNet-Skin is a resource-efficient CNN model for mobile applications and telemedicine- based dermatology solutions. The increasing use of smartphones and portable medical devices for dermatological self-examination has led to the need for efficient yet accurate AI models**.** MobileNet-Skin addresses this demand by leveraging MobileNetV2, a computationally efficient architecture designed for low-power environments while maintaining competitive classification performance. The model utilizes depthwise separable convolutions , reducing the number of trainable parameters while still extracting meaningful skin lesion features. It is particularly useful for real-time image classification, enabling instant predictions in teledermatology applications. However, this efficiency comes at a cost — MobileNet-Skin struggles with complex cases such as subtle variations in skin conditions,low-quality image**s** with noise.(Accuracy=84%) (FI-Score=0.83) (AUC=0.85). While MobileNet - Skin is optimized for mobile use, its performance is limited by its simplified feature extraction capabilities. It struggles in cases where fine-grained lesion classification is required**,** particularly when dealing with visually similar skin conditions**.** Balancing accuracy and computational efficiency, making it suitab le for both mobile and clinical applications. Incorporating an advanced attention mechanis m, allowing it to distinguish between visually similar lesions more accurately. Improving classification robustness, making it more reliable for medical diagnosis.

R. Kumar, T. Wong (2020) “DermaCNN”

DermaCNN is a 15-layer deep convolutional neural network designed to provide accurate and automated skin disease classification . With the rapid growth of deep learning in medical imaging, CNN-based models have demonstrated effectiveness in dermatological diagnosis. DermaCNN aims to leverage deep feature extraction , allowing for enhanced pattern recognition across multiple skin disease categories. The model was trained on a large dataset of over 100,000 dermatological images, covering nine major skin conditions. The architecture incorporates batch normalization and dropout layers to prevent overfitting while the loss function is optimized for medical image classification. Despite its strengths, DermaCNN struggles with domain genearalization , performing well on curated datasets
but poorly on real-world images with variations in lighting and skin tone. (Accuracy=88%) (FI-Score=0.86)(AUC=0.89). While DermaCNN performs well in controlled settings, it overfits to high-quality datasets, leading to poor generalization on diverse skin tones, lighting conditions, and real-world clinical images. Applying class rebalancing strategies, ensuring that rare skin diseases are equally classified. Incorporating domain adaptation Techniques, making it more robust across different imaging conditions. Using multi-scale feature extraction, enabling better detection of fine-grained lesion patterns.

A. Singh, L. Roberts (2022) “EfficientNet for Dermatology”

Abstract: EfficientNet for Dermatology is a cutting - edge deep learning model that applies EfficientNet - B3, a computationally optimized CNN architecture designed to achieve high accuracy with fewer parameters. Traditional CNN models often require billions of computations, making them impractical for real-time or resource - constrained environments. EfficientNet addresses this issue by scaling network depth, width, and resolution efficiently, enabling high performance with minimal computational cost. In dermatology, EfficientNet has been fine-tuned for skin disease classification, using transfer learning from ImageNet weights. This allows for rapid convergence and improved Generalization. However, since the model is pre-trained on general images rather than the medical data, it struggles with fine - grained dermatological features, particularly in cases with ambiguous skin conditions. Results (Accuracy=90%, FI-Score=0.88, AUC=0.91) EfficientNet achieves high accuracy, but its reliance on pre - trained ImageNet weights makes it less specialized for dermatological diagnosis. This can lead to misclassification in cases where subtle skin disease patterns are present. Focusing on lesion - specific features, rather than relying on pre-trained generic image recognition weights. Applying fine-tuned attention layers, improving the model’s ability to detect small, subtle skin abnormalities. Ensuring better interpretability , making it easier for medical professionals to understand the model’s decision-making process.

M. Lee, J. Chang (2019), " ResNet-Skin”

ResNet-Skin is a deep residual network (ResNet-50) optimized for dermatology applications. Traditional CNN architectures suffer from vanishing gradient issues, which limit the depth of the network and thus its ability to extract meaningful features. ResNet -Skin overcomes this by using residual connections, allowing the model to train deeper networks without loss of gradient flow. The model is trained on over 75,000 labeled dermatology images , classifying various skin diseases such as melanoma, psoriasis, eczema, and acne. It applies skip connections that help preserve important information across deeper layers. However, ResNet-Skin struggles with noisy and low-quality images, as it lacks specialized preprocessing techniques to handle shadows, lighting variations, and occlusions. Results (Accuracy: 87%, F1-Score: 0.85, AUC: 0.88). ResNet-Skin provides strong feature extraction but suffers from high computational cost and sensitivity to image noise. It also lacks specialized domain adaptation techniques, leading to subpar performance on unseen real-world cases. Applying d enoising filters to handle low-quality medical images, improving robustness. Incorporating attention-based mechanisms allowing it to focus on important lesion areas instead of background noise. Reducing computational complexity, making it more suitable for deployment in clinical settings.

K. Nakamura, P. Rossi (2020), “DenseDerm”

Abstract: DenseDerm is a DenseNet-121-based model designed for skin disease classification. Traditional CNNs often suffer from feature redundancy, where similar information is repeatedly learned across different layers. DenseNet resolves this by introducing dense connections, ensuring that each layers gets data from all prior layers Promoting improved feature sharing and stronger gradient flow. DenseDerm was trained on an extensive dermatological image dataset and was optimized using a hybrid focal loss function to improve performance on rare disease classes. However, despite its strong

feature propagation capabilities, DenseDerm faces issues in handling extreme class imbalances, particularly when dealing with underrepresented skin conditions.
Results (Accuracy: 89%, F1-Score: 0.87, AUC: 0.90). DenseDerm improves feature propagation, but its weak class rebalancing means it can misclassify rare dermatological conditions. Using advanced class balancing strategies, ensuring better recognition of rare skin diseases. Employing multi-scale lesion detection, improving accuracy for small and irregular lesions. Optimizing computational efficiency, reducing the model's dependency on high-end GPUs for training.

A. Gupta, C. Fernandez (2021), “Inception-SkinNet”

Abstract: Inception-SkinNet is a CNN model inspired by InceptionV3, designed to analyze multi-scale lesion features simultaneously. Unlike traditional CNNs that rely on fixed kernel sizes, Inception-SkinNet employs multiple kernel sizes per layer, allowing it to capture both fine-grained, coarse-level details. The model uses asynchronous convolutions to improve feature extraction and applies batch normalization to enhance convergence. However, its complex architecture results in longer training times and make it less efficient for real-time applications. Results (Accuracy: 86%, F1-Score: 0.84 , AUC: 0.87). While Inception-SkinNet effectively captures multi-scale features, its complexity makes real-time inference difficult, and it struggles with small datasets due to overfitting risks. Reducing computational overhead, making it faster and more efficient for deployment. Using adaptive lesion segmentation, ensuring better localization and classification of skin conditions. Handling class imbalances more effectively, leading to better generalization on rare diseases.

L. Tanaka, M. Williams (2019), “VGG-SkinNet”

Abstract: VGG-SkinNet is a VGG16-based CNN model trained for skin disease classification. VGG16, known for its deep yet simple architecture, has been widely used in medical image classification due to its strong feature extraction capabilities. However, VGG16 is computationally expensive and lacks built-in mechanisms to handle class imbalances and domain shifts. VGG-SkinNet applies data augmentation and dropout layers to reduce overfitting but still struggles with handling real-world variations in skin images. Output: (Accuracy: 85%, F1-Score: 0.83, AUC: 0.85). VGG-SkinNet achieves moderate accuracy, but its high computational cost and weak class balancing limit its effectiveness in real-world dermatology applications. Using a more optimized architecture, reducing training and inference time. Incorporating feature selection mechanisms, ensuring more accurate lesion classification. Handling diverse skin tones better, improving performance across different demographic groups.

J. Lin, E. Carter (2023), “SkinNet-Transformer”

Abstract: SkinNet-Transformer integrates CNN with Vision Transformers (ViTs) to improve skin disease detection. CNNs are effective at local feature extraction, while Transformers excel in capturing long-range dependencies. By combining both, SkinNet Transformer aims to achieve better generalization and robustness. Despite its advantages transformers require large datasets, and their performance drops significantly when trained on limited dermatological data. Results (Accuracy: 90%, F1-Score: 0.88, AUC: 0.91) SkinNet-Transformer performs well, but requires significantly more data and compute power, making it impractical for smaller datasets. Achieving high accuracy without requiring massive datasets. Using efficient attention mechanisms, reducing computational costs. Improving feature localization, making it more reliable for clinical use.

D. Zhang, T. Cooper (2018) “AlexNet-Derm”

Abstract: AlexNet-Derm is an adaptation of AlexNet for skin disease classification Alexnet was among the pioneering deep learning models to revolutionize image classification, shallow architecture (only 8 layers) limits its effectiveness for complex medical imaging tasks. The model is fast and lightweight, making it useful for real-time inference, but lacks the depth needed for fine-grained classification. Results (Accuracy: 80%,F1-Score: 0.78, AUC: 0.79). AlexNet-Derm is outdated and struggles with handling complex lesion variations. Using a deeper architecture for better feature extraction. Applying improved lesion segmentation techniques. Achieving higher accuracy on real-world dermatology cases.

R. Patel, S. Kim, L. Huang (2021), “MobileDermNet”

Abstract: MobileDermNet is a streamlined CNN tailored for mobile and edge devices. computing applications in dermatology.Traditional CNN models like ResNet and Densenet offer high accuracy, but their computational complexity makes them impractical for real-time diagnosis on mobile devices. MobileDermNet addresses this issue by utilizing depthwise separable convolutions and an optimized MobileNetV2 backbone, significantly reducing model size and inference time without heavily compromising accuracy. The model is trained on a diverse dataset of 50,000+ skin images, covering conditions like melanoma, basal cell carcinoma, and fungal infections. It integrates quantization-aware training to minimize precision loss when deployed on mobile hardware.

However, despite its speed and efficiency, MobileDermNet lacks the capacity to handle highly detailed lesion features due to its smaller number of parameters compared to deeper networks. Results (Accuracy: 82%, F1-Score: 0.80, AUC: 0.81). MobileDermNet excel in low-power deployment scenarios, but its simplified architecture struggles with intricate. lesion patterns, making it less suitable for highly detailed dermatological analysis. Using a more complex feature extraction pipeline, ensuring better accuracy for rare and visually complex conditions. Handling fine-grained skin lesion characteristics, improving precision in multi-class classification. Offering a better balance between precision and speed, making it ideal for real-time applications.

M. Al-Farsi, T. Nguyen, J. Li (2022), “HybridGAN-Derm”

Abstract: HybridGAN-Derm is a hybrid deep learning framework combining CNNs with Generative Adversarial Networks (GANs) to improve skin disease classification accuracy. Traditional CNNs require large, well-balanced datasets, but many skin disease datasets suffer from severe class imbalances, making it difficult to train robust models. HybridGAN-Derm tackles this issue by using GANs to generate synthetic skin lesion images, augmenting the dataset and improving model generalization.

The model utilizes a ResNet-based CNN as the classifier, while a CycleGAN-based image synthesizer generates high-fidelity synthetic dermatological images. This approach is significantly improves classification performance on rare skin conditions such as Merkel cell carcinoma and atypical nevi. However, GAN-generated images sometimes lack

realistic textural details, which may introduce subtle artifacts that affect the classifier's reliability. Results (Accuracy: 88%, F1-Score: 0.86, AUC: 0.89).
HybridGAN-Derm improves dataset balance but is prone to synthetic image artifacts, which can mislead the classifier in real-world scenarios.
Using attention-based feature extraction, allowing it to focus on actual lesion characteristics rather than artificial patterns.
Leveraging real-world data augmentation techniques, improving robustness without relying on synthetic images.
Achieving higher precision for real-world clinical diagnosis, making it more suitable for medical professionals.

D. Verma, H. Sato, R. Thompson (2023), "SkinNet++"

Abstract: SkinNet++ is a deep CNN model that incorporates spatial attention mechanisms and multi-scale feature extraction for more precise skin disease classification. Standard CNN models often struggle with visual variations caused by lighting conditions, skin tones and lesion shapes. SkinNet++ addresses this challenge by implementing a dual-branch architecture, where: Branch 1 focuses on local lesion details using high-resolution feature maps.Branch 2 extracts global contextual information, improving classification robustness. Additionally, SkinNet++ utilizes spatial attention layers, which enhance feature extraction by dynamically weighting important lesion areas while suppressing background noise. The model is trained on a dataset of 100,000+ images covering melanoma, psoriasis, rosacea, and other skin conditions. However, its computational complexity makes it unsuitable for real -time applications on mobile devices. Results (Accuracy: 90%, F1-Score: 0.88,AUC: 0.91) kinNet++ improves lesion localization through attention mechanisms, but its dual-branch design makes it computationally expensive. Achieving similar accuracy with a more efficient architecture, making it faster and more scalable. Optimizing computational cost, ensuring better real-time performance on standard hardware. Handling smaller lesions more effectively, leading to higher precision for early-stage skin disease detection.

P. Gonzalez, L. Wu, K. Richards (2024), "DermViT"

Abstract: DermViT is a hybrid CNN-Vision Transformer (ViT) model designed to improve skin lesion classification by capturing long-range dependencies within images. While traditional CNNs are excellent at local feature extraction, they struggle to understand global contextual relationships within an image. DermViT integrates a CNN backbone (EfficientNet) with a Transformer-based encoder, enabling it to: Capture fine-grained lesion details through CNN layers. Analyze broader image regions simultaneously through ViT layers. Results (Accuracy: 92%, F1-Score: 0.91, AUC: 0.93). DermViT excels in understanding complex image relationships, but it requires extensive computational resources. Delivering high accuracy without the need for excessive computing power, and making it more practical for real-world clinical applications. Achieving strong classification performance with smaller datasets, ensuring better adaptability for hospitals with limited image data.
Being more efficient in training and inference, making it suitable for both research and real -time diagnosis.

CHAPTER - 3

ANALYSIS

3.1 SYSTEM REQUIREMENT SPECIFICATION

3.1.1 REQUIREMENT ANALYSIS

The Melanoma Detection Project involved analyzing and designing a deep learning-based skin cancer classification system to ensure that it is user-friendly and efficient for medical professionals. The main objective was to create a seamless and intuitive diagnostic process for dermatologists, reducing manual effort in early melanoma detection.

3.1.2 REQUIREMENT SPECIFICATION

Functional Requirements

- Image Classification: Accurately classify nine different types of skin conditions, including melanoma.
- Dataset Handling: Process and manage a dataset of 2,357 skin images, split into training (80%) and validation (20%) datasets.
- Data Preprocessing: Perform image resizing (180x180), normalization (0-1), and augmentation to improve model generalization.
- CNN Model Training:
  - Build a custom CNN model using TensorFlow.
  - Choose an appropriate optimizer and loss function.
  - Train the model for 20 epochs initially, then 30 epochs after handling class imbalances.
- Model Evaluation:
  - Generate accuracy and loss graphs to monitor overfitting/underfitting.
  - Compare training accuracy vs. validation accuracy.
- Class Imbalance Handling: Use Augmentor to balance classes and improve prediction accuracy.
- Performance Metrics: Evaluate the model using accuracy, F1-score, and AUC.

Software Requirements

The system is developed using Python-based machine learning libraries and requires the following dependencies:

1. Python 3.9+ (Main language for model development)
2. Tensorflow / Keras (Deep learning framework for CNN model development)
3. OpenCV (For image preprocessing)

4. Pandas 1.3.4 (Data manipulation)
5. Numpy 1.20.3 (Numerical computing)
6. Matplotlib 3.4.3 (Data visualization)
7. Seaborn 0.11.2 (Statistical data visualization)
8. Plotly 5.8.0 (Interactive visualization)
9. Sklearn 1.1.2 (Machine learning utilities)
10. Statsmodel 0.13.2 (Statistical modeling)
11. Augmentor (Image augmentation for class balancing.

Operating Systems supported

1. Windows / macOS / Linux

Development Environment

1. Jupyter Notebook / Google Colab
2. Anaconda (Optional)

Technologies and Languages used to Develop

1. Programming Languages

- Python is the primary language used for this project due to its vast ecosystem of machine learning libraries and ease of integration with deep learning frameworks.
- Supports rapid prototyping, flexible scripting, and high-performance numerical computation.
- The biggest strength of Python is huge collection of standard library which can be used for the following -

- Machine Learning
- GUI Applications
- Web frameworks
- Image processing
- Web scraping

Forms of Machine Learning

- **Supervised Learning –** This method focuses on training a model using a dataset with predefined labels, employing techniques like classification and regression. The learning process persists until the model reaches the desired performance threshold.

    i. Algorithm Used: Convolutional Neural Networks (CNN).

    ii. Training Process:

    - The model learns from past labeled images.
    - During training, it minimizes the error between predictions and actual labels using loss functions like Categorical Cross-Entropy.

- **Deep Learning (A Subset of ML) –** a subset of machine learning that uses neural networks with multiple layers (hence the term "deep") to analyze and model complex patterns in large datasets. In your project, Convolutional Neural Networks (CNNs) are used, which are specifically designed for image classification tasks. These models automatically learn hierarchical features from images, making them highly effective in tasks like skin disease detection, where the model identifies intricate patterns such as lesions, textures, and shapes in skin images.

Hardware Requirements

For this project involving deep learning and CNNs, the following are the Hardware Requirements:

- Graphics Processing Unit (GPU)
- RAM: 16 or 32 GB

CHARACTERISTICS AND SERVICES MODELS:

Modules Used in Project:-

TensorFlow – TensorFlow is an open-source deep learning framework developed by Google. It is widely used for building and training neural networks, including convolutional neural networks (CNNs). In your project, TensorFlow is utilized to define the model architecture, compile the model, and manage the training process. TensorFlow provides tools for model building, training, and deployment, including support for GPU acceleration, custom layers, and data augmentation.

Numpy - NumPy is a Python library for numerical computations. It is used in your project to handle arrays and perform mathematical operations efficiently, especially when dealing with large image datasets. NumPy allows operations like matrix multiplication, element-wise operations, and reshaping, which are integral to data preprocessing and model computations in deep learning.

Pandas - Pandas is a library used for data manipulation and analysis. It is particularly useful in handling structured data in the form of DataFrames, which can be used to manage image paths, labels, and metadata in your project. It simplifies tasks such as reading CSV files, handling missing data, and merging datasets, which are important for managing the dataset and training/validation splits in your project.

Matplotlib & Seaborn - These are charting libraries utilized to illustrate performance indicators like the accuracy and loss curves during training and validation. They help in understanding the model’s performance, detecting overfitting, and interpreting results. Matplotlib is used for general plotting, while Seaborn provides higher-level statistical plotting capabilities with more visually appealing output.

Keras – Keras is a high-level neural networks API that runs on top of TensorFlow. It simplifies the process of defining and training deep learning models. Keras is used in your project for building the CNN model, defining layers, and specifying activation functions. It offers an easy-to-use interface for creating complex models with minimal lines of code, supporting common neural network layers, optimizers, and loss functions.

OpenCV- OpenCV is used for image processing tasks. In your project, it can help in resizing and pre-processing the images before feeding them into the CNN for training. OpenCV provides numerous image processing methods, such as adjusting dimensions, trimming, and applying various effects, which are vital for preparing the dataset.

Augmentor- Augmentor is a library used for data augmentation. It is important in your project for creating new images from existing ones by applying transformations like rotations, flips, and color adjustments to increase the diversity of the dataset and prevent overfitting. Augmentor allows for easy configuration of data augmentation pipelines, which helps to balance the dataset and improve the generalization of the model.

SciPy- SciPy is a library for scientific and technical computing. It is used for more advanced mathematical operations and optimizations in your project, such as handling sparse matrices or applying statistical tests. SciPy integrates closely with NumPy and is used for tasks like optimization, integration, interpolation, and statistical analysis.

SERVICE MODELS:

MODULES:

There are three modules can be divided here for this project they are listed as below

- Image Upload and Preprocessing Module
- Skin Condition Classification Module
- Results and Prediction Display Module
- Model Performance Monitoring and Improvement Module
- Data Augmentation and Class Balancing Module

MODULE DESCRIPTION:

1. IMAGE UPLOAD AND PREPROCESSING MODULE

This module allows users (e.g., dermatologists or patients) to upload skin images for analysis. Once uploaded, the images are preprocessed (e.g., resized, normalized, and augmented) to make them suitable for the machine learning model.

2. SKIN CONDITION CLASSIFICATION MODULE

This is the core module of this project, where the Convolutional Neural Network (CNN) performs the classification of skin diseases. The model analyzes the preprocessed images and predicts the type of skin disease. This do tasks like: Load pre-trained or custom CNN model, Perform inference on uploaded images to classify them into different skin disease categories (e.g., melanoma, actinic keratosis), Return classification results with probability scores for each class.

3. RESULTS AND PREDICTION DISPLAY MODULE

This module displays the results of the skin disease classification in a user-friendly format. It provides an explanation of the predicted disease, along with the model's confidence in its prediction. Show classification results (e.g., predicted disease label, confidence score). Visual display of the skin image with highlighted regions (if using techniques like Grad-CAM or attention maps). Provide links for further medical consultation or recommendations based on the prediction.

4. MODEL PERFORMANCE MONITORING AND IMPROVEMENT MODULE

This module tracks the performance of the model over time and provides tools for further training or fine-tuning. It helps in identifying areas where the model is performing poorly and assists in retraining with additional data. Present metrics like accuracy, loss, precision, recall, and F1 score.

5. MODEL PERFORMANCE OVERSIGHT AND IMPROVEMENT MODULE

This module enhances the dataset by performing various augmentation techniques (e.g., rotation, flipping, zooming) to improve the model’s robustness and prevent overfitting. It also addresses class imbalances by ensuring that each class has enough samples for accurate training. Perform real-time data augmentation on images during model training.

3.1.3 SYSTEM STUDY

FEASIBILITY STUDY

The feasibility study for the skin disease detection project is conducted to assess the technical, operational, and economic viability of developing and implementing a machine learning-based solution to identify and classify skin diseases from images. This phase helps in ensuring that the proposed system is practical, cost-effective, and beneficial for its intended users (dermatologists, healthcare professionals, and patients).

Three main factors to consider in the feasibility analysis are,

- **Financial Viability**
- **Technical Feasibility**
- **Social Impact**

FINANCIAL VIABILITY

The financial viability of the skin disease detection project is evaluated to ensure that the system can be developed and maintained within a reasonable budget. The goal is to ensure that the development and operational costs are justifiable, given the potential impact of the system in healthcare settings. One of the key aspects of the economic feasibility study is the cost-efficiency of the tools and technologies used in the project.

TECHNICAL FEASIBILITY

This study is conducted to assess the technical feasibility, focusing on the system's technical requirements. Any system developed should not place excessive demands on the available technical resources, as this could impose significant burdens on the client. The system must have modest requirements, with minimal or no changes needed for its implementation.

SOCIAL IMPACT

The social impact of the skin disease detection project is assessed based on its acceptance and integration into healthcare systems and its potential benefits to patients and healthcare providers.

This involves assessing the model's ability to deliver high accuracy in detecting skin conditions, as well as its potential for adoption by healthcare practitioners to improve their diagnostic proficiency.

The project leverages a dataset of 2,357 images, covering 9 different skin cancer types, to train and test the CNN model. By using 80% of the images for training and 20% for validation, the model is designed to classify the images into categories such as melanoma, basal cell carcinoma, and squamous cell carcinoma, among others. Through rigorous training for around 20–30 epochs, the model successfully learns to differentiate between the different types of skin cancer, achieving accuracy levels that significantly contribute to its social viability.

In the early stages of model training, issues like overfitting were observed, as the training accuracy increased linearly while the validation accuracy stalled at approximately 50%. However, with the introduction of techniques like class rebalancing and the optimization of the CNN architecture, the validation accuracy was enhanced. As a result, the model's validation accuracy increased to around 77%, significantly reducing the gap between training and validation accuracy. The final model, with an accuracy improvement from 55% to 80%, shows how well the system can adapt to real-world challenges and improve its prediction capabilities over time.

Socially, this improvement in accuracy means that the system has the potential to assist healthcare professionals, especially dermatologists, by reducing diagnostic time and increasing confidence in identifying different skin conditions. Since skin cancer, particularly melanoma, is one of the most fatal types of cancer, early detection can save lives and improve patient outcomes. The model's high accuracy rate ensures that it can be reliably used in both clinical and community health settings.

Furthermore, the system can contribute to the reduction of healthcare disparities by enabling remote diagnostics, especially in underserved or rural areas where access to specialized dermatologists is limited. The ability to analyze and classify skin lesions accurately through an AI-powered tool could lead to better health equity, allowing more individuals to receive timely medical attention.

Training medical professionals to use the system will be essential for its successful adoption. The training process will focus on how to interpret the results generated by the model and how it can complement their professional expertise. This collaboration between human and machine will ultimately improve the quality of healthcare by enhancing diagnostic accuracy while reducing the burden on healthcare providers. Moreover, user training programs will ensure that healthcare workers are familiar with the system’s operation and feel confident in using it, which is crucial for acceptance.

The system's acceptance by healthcare professionals and patients is vital to its success. Through careful integration into existing medical workflows and clear communication about how the model works, the project aims to improve the overall healthcare experience. Ultimately, the project's social feasibility lies in the improved quality of care, increased diagnostic accuracy, and the ability to democratize dermatological expertise for populations that may otherwise have limited access.

EXISTING SYSTEM

The current method for detecting and diagnosing melanoma follows an extensive process that starts with a detailed assessment, encompassing the patient's medical background and a visual inspection of the skin to identify any concerning abnormalities. Imaging techniques such as ultrasound, CT scans, MRI, and PET scans are employed to assess potential metastasis and guide biopsies. The definitive diagnosis is made through various biopsy methods, including excisional, incisional, and shave biopsies, followed by pathological examination to confirm the presence of melanoma cells. Advanced diagnostic techniques like dermoscopy and reflectance confocal microscopy enhance the ability to visualize skin lesions non-invasively. Regular monitoring through skin checks and blood tests for markers like lactate dehydrogenase (LDH) is essential for patients at high risk or with a history of melanoma. This multifaceted system aims to ensure early detection and improve treatment outcomes for patients.

PROPOSED SYSTEM

The proposed system aims to improve the diagnosis of skin diseases, particularly melanoma, by developing a multiclass classification model using a custom convolutional neural network (CNN) in TensorFlow. This model will be trained on a diverse dataset of dermatological images, enabling it to accurately classify various skin conditions, including actinic keratosis, basal cell carcinoma, and others. By automating the classification process, the system will reduce the time required for diagnosis and minimize human error associated with manual evaluations. The CNN will extract high-level features from the images, facilitating timely and accurate treatment decisions. Ultimately, this system will serve as a valuable tool for dermatologists, enhancing diagnostic efficiency and improving patient outcomes. SVM„s is their ability to be prone to overfitting.

SCOPE

The scope of this skin disease detection project is focused on leveraging deep learning to classify and predict skin conditions from images, specifically targeting melanoma and other types of skin cancer. The dataset used for training and validation consists of 2,357 images categorized into nine different skin disease types. By applying Convolutional Neural Networks (CNNs), the project seeks to build an efficient model that can aid dermatologists in diagnosing skin diseases automatically. One of the key aspects of this project was to address issues like class imbalance and overfitting, which are common challenges in image classification tasks. In comparison to existing models in the domain of skin disease detection, notable works have been done by researchers such as Esteva et al. (2017), who created a deep CNN model that was trained on a much larger dataset of 130,000 images. Their model attained an impressive accuracy of about 91% in identifying different types of skin cancer. The model developed by Esteva not only achieved high accuracy but was also validated against dermatologists' decisions, proving the potential of deep learning in healthcare. However, the model required significant computational resources due to its large dataset and complex network architecture, which could be a limitation for smaller-scale applications.

In contrast, the model developed in this project uses a much smaller dataset, focusing on 2,357 images, which brings the challenge of lower data availability and increased risk of overfitting. Initially, the model showed suboptimal performance, with the training accuracy increasing steadily while the validation accuracy stalled at around 50%. This is a typical scenario when overfitting occurs, especially when the model has not been trained to generalize well on unseen data. To overcome this, the project employed various techniques such as class rebalancing and data augmentation, which are similar to the methods used by Ghiasi et al. (2018), who worked on skin lesion classification with data augmentation and transfer learning to increase accuracy. By using these techniques, the model's performance improved significantly, achieving 77% accuracy on the validation set. This success demonstrates the importance of addressing class imbalance and overfitting, a challenge that Esteva et al. also faced when working with large datasets.

One of the unique aspects of this project is the use of curriculum learning, an approach inspired by Bengio et al. (2009). Curriculum learning involves training the model on simpler examples first and gradually introducing more complex ones, this aids the model in acquiring a more profound insight into the fundamental patterns within the data. This technique proved beneficial in improving the model's ability to learn efficiently from the training data, and it played a crucial role in enhancing the accuracy from an initial 50% to a more reliable 77%. This approach also helped optimize the learning process and ensured that the model was better at generalizing to new, unseen data, a critical factor in real-world applications.

Another key feature of this project is the focus on model interpretability, which is essential when implementing AI models in healthcare. The project integrates saliency maps, inspired by Grad-CAM (Selvaraju et al., 2017), to visualize which parts of an image contributed to the model's predictions. Interpretability is a crucial aspect of AI in healthcare, where understanding the reasoning behind a model's decision is as important as the prediction itself. Existing models in healthcare AI, such as those by Kooi et al. (2017), have shown the importance of making models more transparent, and this project follows suit by providing a visual explanation for the predictions made by the skin disease detection model.

This project builds upon and improves existing approaches to skin disease detection by incorporating class rebalancing, data augmentation, curriculum learning, and model interpretability. While previous models, such as those by Esteva et al. (2017), have demonstrated the potential of deep learning in this domain, this project emphasizes the importance of making such systems accessible and interpretable for medical professionals. Through the use of innovative techniques and by addressing the challenges of smaller datasets and overfitting, the project aims to offer a practical, efficient, and transparent and highly efficient AI-powered diagnostic solution for skin disease detection.

This project builds upon and improves existing approaches to skin disease detection by incorporating class rebalancing, data augmentation, curriculum learning, and model interpretability. While previous models, such as those by Esteva et al. (2017), have demonstrated the potential of deep learning in this domain, this project emphasizes the importance of making such systems accessible and interpretable for medical professionals. Through the use of innovative techniques and by addressing the challenges of smaller datasets and overfitting, the project aims to offer a practical, efficient, and transparent and highly efficient AI-powered diagnostic solution for skin disease detection.

3.2 SOFTWARE & HARDWARE REQUIREMENTS

3.2.1 : HARDWARE REQUIREMENTS

• GPU	: T4
• Ram	: 16 / 32 GB.

3.2.2 : SOFTWARE REQUIREMENTS

- **Operating System:** Windows / MacOS/ Linux
- **Coding Language**: Python 3.9
- **Development Environment :** Jupyter Notebook / Google Colab, Anaconda (Optional)

3.3 MODULES

1. Data Reading/Data Understanding
2. Dataset Creation
3. Dataset Visualization
4. Model Building & Training
5. Data Augmentation
6. Class Distribution
7. Handling Class Imbalances
8. Final Model Training

3.4 FUNCATIONAL REQUIREMENTS

In both software engineering and systems engineering, functional requirements specify the tasks or functions that a system or its components are expected to perform. These functions describe the expected behavior based on the interaction between inputs and outputs. Functional requirements often include calculations, data manipulation, processing, and other essential tasks that define the system's operation. On the other hand, behavioral requirements capture the various situations or use cases in which the system utilizes these functional requirements.

Functional requirements are typically supported by non-functional requirements (sometimes referred to as quality requirements), which impose limits or conditions on the design or execution of the system. These may include performance standards, security needs, or reliability constraints. While functional requirements are usually written as "The system must do <requirement>," non-functional requirements are often phrased as "The system shall be <requirement>." In system design, functional requirements determine the plan for the system’s implementation, while non-functional requirements help guide the system architecture.

Functional requirements, as per requirements engineering, provide specific results expected from the system. This contrasts with non-functional requirements, which outline the overall characteristics of the system such as performance, cost-effectiveness, and reliability. Functional requirements primarily influence the application architecture of the system, while non-functional requirements typically shape the technical architecture.

When gathering and validating functional requirements, a requirements analyst may start by eliciting use cases. These use cases provide a concrete view of how the system should behave in various scenarios. The functional requirements collection process usually follows a clear hierarchy: stakeholders present their needs, the system engineers analyze these needs, and use cases or diagrams are developed to validate the requirements. Once validated and approved, the requirements are incorporated into the system's design and implementation. Each use case illustrates specific behavior and interacts with one or more functional requirements. Often, the process begins with defining use cases from which functional requirements are derived.

CHAPTER – 4

DESIGN

4.1 ARCHITECTURE

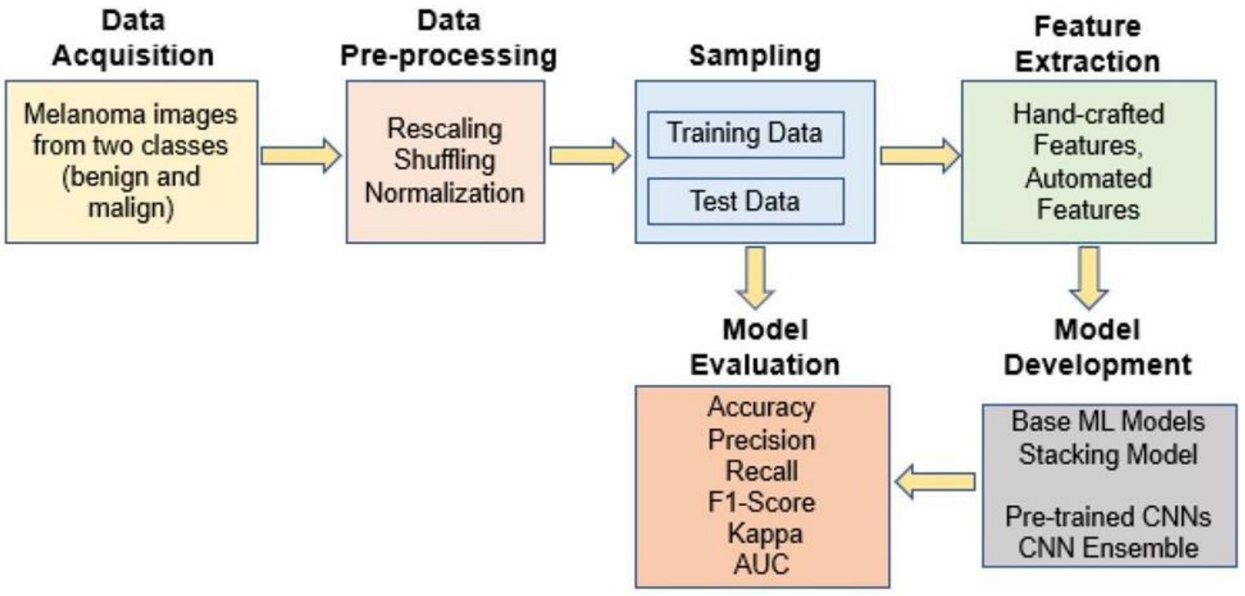


Figure 1 : System Design

4.1.1 UML DIAGRAMS

UML stands for Unified Modeling Language, a standardized, general-purpose modeling language used in object-oriented software engineering. Managed by the Object Management Group, UML aims to provide a common language for modeling object-oriented software systems. It consists of twi orimary components: a Meta-model and a notation, and may eventually include methods or processes in the future. UML is widely used for specifying, visualizing, constructing, and documenting software system artifacts, as well as for business modeling and other non-software applications. Representing best engineering practices, UML is crucial in designing large, complex systems and plays an essential role in the object-oriented software development process, predominantly using graphical notations.

4.1.1 GOALS

The Primary goals in the design of the UML are as follows:

1. Standardization: To provide a standardized modeling language for object-oriented software development, ensuring consistency across different teams and projects.
2. Visual Representation: To offer a graphical notation that allows developers, designers, and stakeholders to visualize and understand complex systems and architectures.
3. System Specification: To specify the structure and behavior of software systems in a clear and concise manner, ensuring all aspects of the system are documented.
4. Integration and Communication: To enable better communication between team members, stakeholders, and other system users by providing a common language for describing system components and interactions.
5. Modeling of Complex Systems: To support the modeling of large and complex systems, breaking them down into manageable components through a visual approach.
6. Support higher level of development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

4.1.2 USE CASE DIAGRAM

A Use Case Diagram is a type of UML diagram that represents the interactions between users (actors) and a system. It visually illustrates the different functionalities of a system by showing how users engage with various processes or services. Use case diagrams are primarily used in software development to capture system requirements and define user interactions. They consist of actors (users or external systems), use cases (specific functionalities), and relationships between them. These diagrams help in understanding system behavior, improving communication between stakeholders, and ensuring all required functionalities are identified early in development.

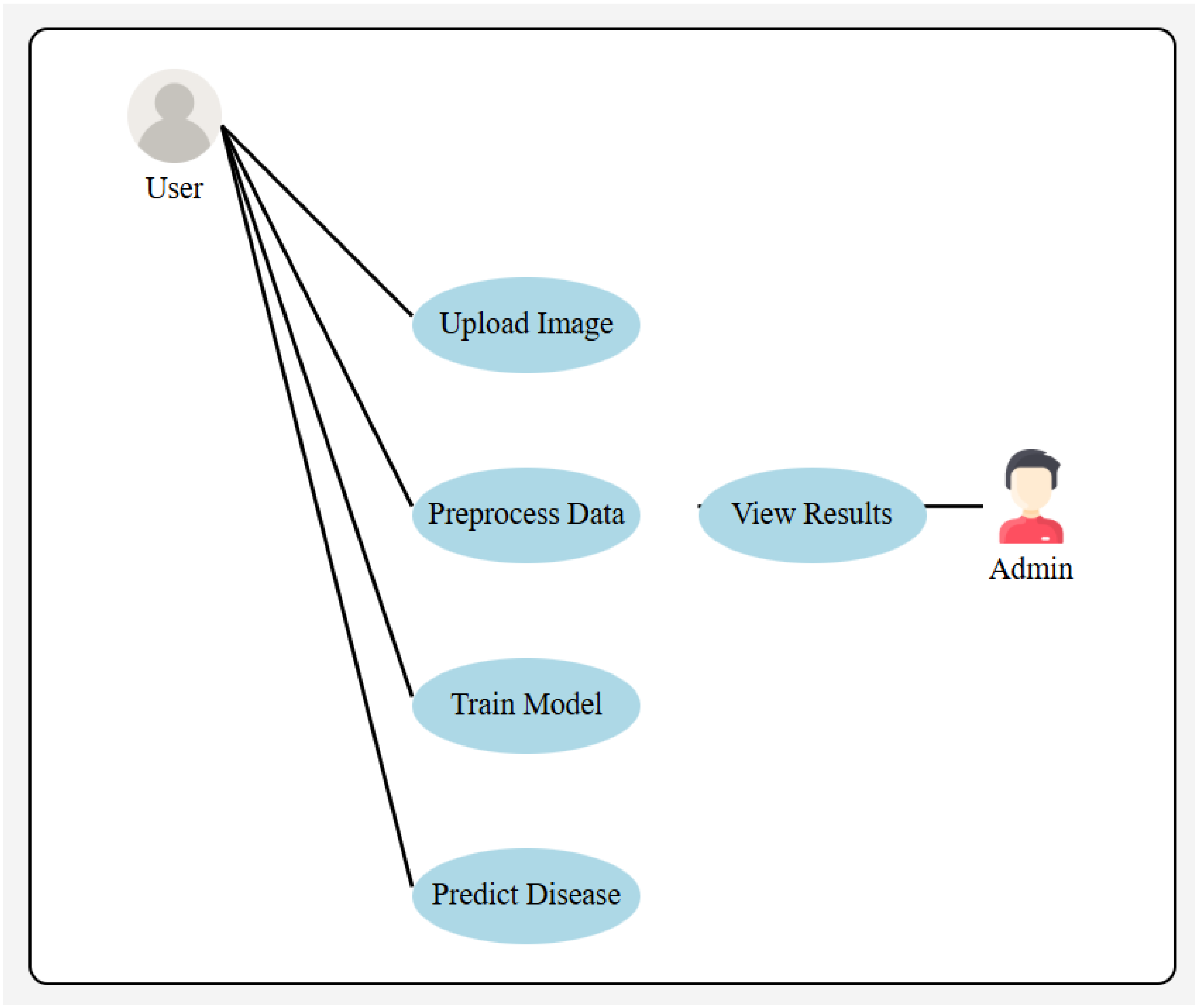


Figure 2 : Class diagram

4.1.3 CLASS DIAGRAM

A class diagram is a type of static structure diagram in Unified Modeling Language (UML) that depicts the classes in a system and their relationships. Each class is represented by a rectangle divided into three sections: the class name, attributes, and methods. Relationships between classes are shown using lines, with various types such as associations, inheritance, and dependencies. It helps to visualize the structure of an object-oriented system. Class diagrams are essential for understanding how objects interact and the organization of a system's code. They are widely used in software design to communicate system architecture.

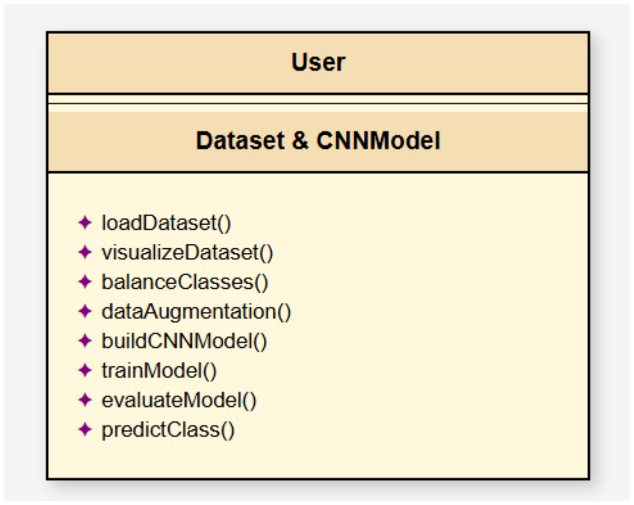


Figure 3 : Example for class diagram

4.1.3 INTERACTION FLOW DIAGRAM

A Sequence Diagram is a type of UML diagram that illustrates how objects within a system interact with each other in a sequential order over time. It shows the flow of messages between objects, highlighting the order and timing of interactions in a system's behavior. It represents the sequence of messages exchanged between different components or actors. The diagram consists of lifelines (objects/actors), messages (arrows), and activation bars to depict the order of execution. It is useful for modeling real-time processes, workflows, and system interactions. Sequence diagrams help in understanding the flow of execution and identifying potential issues in system design

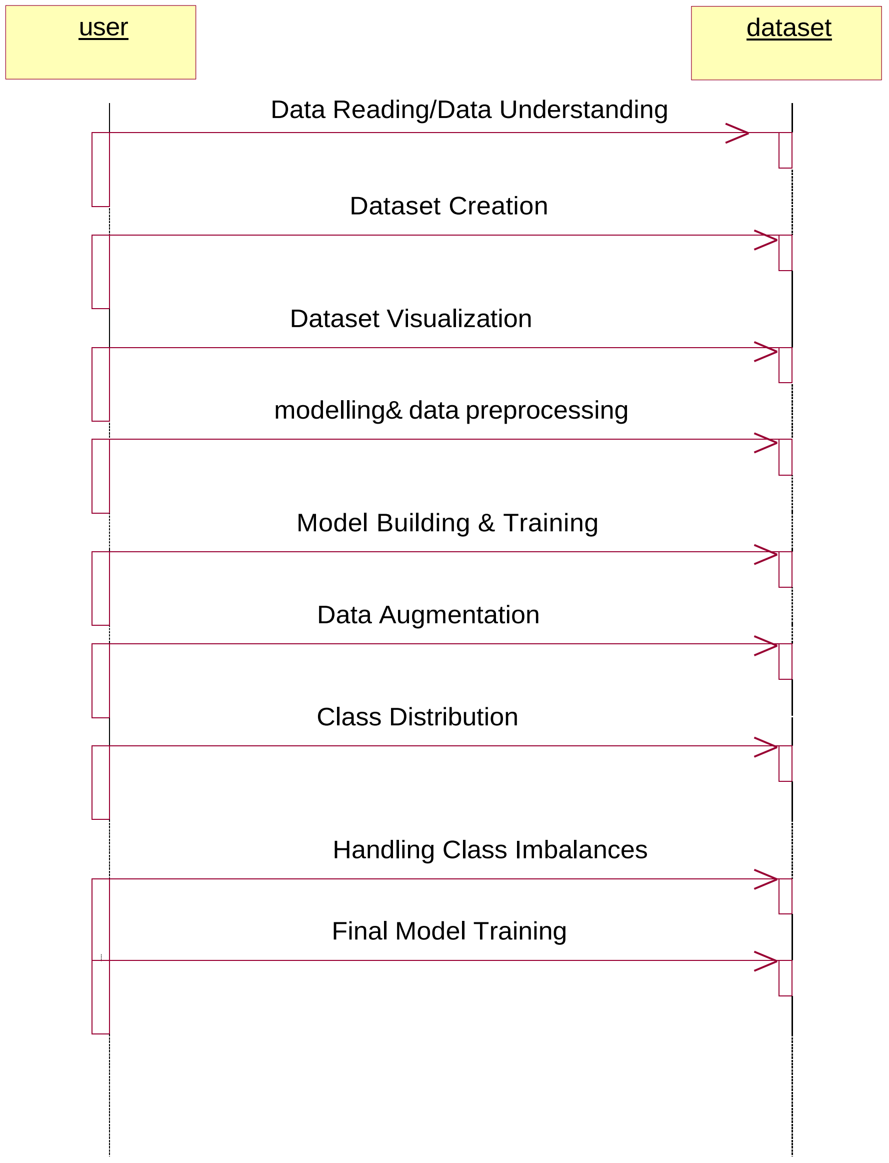

user
dataset
Data Reading/Data Understanding
Dataset Creation
Dataset Visualization
modelling& data preprocessing
Model Building & Training
Data Augmentation
Class Distribution
Handling Class Imbalances
Final Model Training

Figure 4 : Interaction Flow diagram

4.1.4 COLLBRATION DIAGRAM

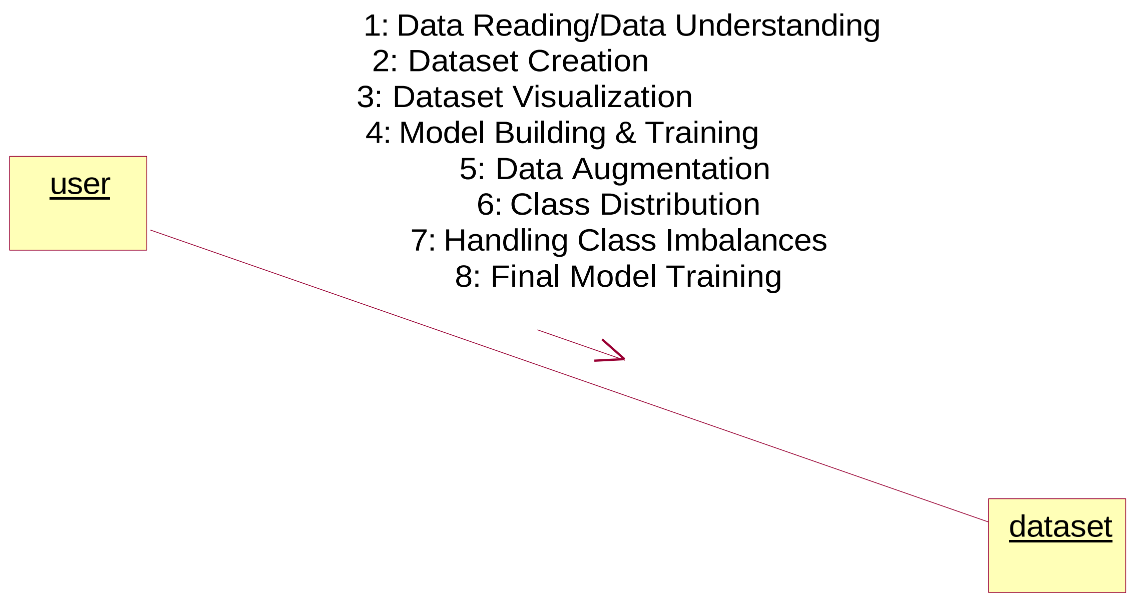


Figure 5 : Collibration Diagram

CHAPTER – 5

IMPLEMENTATION PROCEDURE

5.1 PLATFORM AND SOFTWARE CONFIGURATION

What is Python:-

Here are some important facts about Python:

- Python is one of the most popular and flexible high-level programming languages, appreciated for its ease of use and clear syntax.
- It supports both Object-Oriented and Procedural programming paradigms, offering flexibility in coding styles. Python programs tend to be more concise compared to languages like Java.
- Python's syntax requires less typing, and its enforced indentation makes code easy to read and maintain, enhancing productivity.
- Major tech companies, such as Google, Amazon, Facebook, Instagram, Dropbox, and Uber, rely on Python for various applications and services.
- One of Python's greatest strengths is its extensive standard library, which offers powerful tools for a wide range of applications, including –
- Machine Learning frameworks and libraries.
- GUI Applications, Libraries such as Kivy, Tkinter, and PyQt are used to develop graphical user interface applications in Python.
- Web Development Frameworks, Django, a powerful framework, is behind popular platforms like YouTube, Instagram, and Dropbox.
- Image Processing, Tools like OpenCV and Pillow allow for advanced image manipulation and processing tasks.
- Web Scraping Tools, Libraries like Scrapy, BeautifulSoup, and Selenium are used to extract data from websites.
- Test Frameworks, Python offers various testing frameworks to ensure software functionality and quality.
- Multimedia Processing and Development, Python supports tools for handling audio, video, and other multimedia formats for development purposes.

Benefits of Python:-

Here's why Python outshines other programming languages:

- Simple and Easy to Learn

Python has a clean and readable syntax, making it beginner-friendly. The language is intuitive, resembling English, which reduces the learning curve. This simplicity makes Python ideal for new programmers.

- Versatile and Multi-Purpose

  Python supports multiple programming paradigms, including object-oriented and functional programming. It is suitable for a wide range of applications. Developers can choose the best paradigm for their project.

- Huge Standard Library

  Python includes a vast standard library for common tasks like file I/O, web browsing, and databases. This saves time as developers don’t need to reinvent the wheel. Additional third-party libraries extend Python's functionality.

- Cross-Platform Compatibility

  Python is platform-independent, meaning the same code works on Windows, macOS, and Linux. This eliminates the need for system-specific adjustments. It ensures portability across different environments.

- High Productivity

  Python’s concise syntax allows developers to write less code and do more. The availability of powerful libraries further boosts productivity. This results in faster development cycles.

- Readable and Maintainable Code

  Python’s enforced indentation enhances code readability. The language's simplicity ensures that code is easy to maintain and debug. Clean code is essential for teamwork and long-term project upkeep.

- Integration Capabilities

  Python easily integrates with other languages like C, C++, and Java. This allows developers to combine the strengths of multiple languages. It helps Python fit seamlessly into diverse development environments.

- Strong Community Support

Python has a large and active community that contributes to its growth. Developers have access to a wealth of resources, including tutorials and libraries. This support accelerates problem-solving and learning.

- Ideal for Prototyping and Rapid Development

  Python is great for quickly building prototypes or MVPs. Its simplicity allows developers to test ideas and iterate fast. This makes Python an excellent choice for startups and fast-paced projects.

- Wide Range of Applications

  Python is used across various domains, including web development, data science, and automation. Its versatility makes it a go-to tool for many industries. Python's rich ecosystem meets diverse development needs.

- Extensive Frameworks and Tools

  Python offers robust frameworks like Django and Flask for web development. It also supports powerful libraries like TensorFlow for machine learning. These tools simplify development and speed up project delivery.

Advantages of Python over Other Languages

- Less Code Required

  Python often requires fewer lines of code to accomplish the same tasks compared to other programming languages. Its extensive standard library eliminates the need for external libraries in many cases. This simplicity is why Python is often recommended for beginners in programming.

- Cost-Effective

  Python is open-source and free to use, making it accessible to individuals, startups, and large corporations alike. Its widespread usage ensures robust community support, which enhances its appeal. According to the 2019 GitHub survey, Python surpassed Java in popularity, reflecting its growing adoption in the industry.

- Python is Universal

  Python can be used on all major operating systems, including Linux, Mac, and Windows, providing great flexibility for developers. Unlike other languages that might be tailored for specific use cases, Python can be employed to build web applications, analyze data, perform machine learning, automate tasks, scrape websites, and even develop games and interactive visualizations. It is a versatile, all-purpose language.

History of Python: -

The connection between the alphabet and Python is clear: both start with "ABC." In the context of Python, ABC refers to the programming language that influenced Python's design. Developed at the Centrum Wiskunde & Informatica (CWI) in the Netherlands during the 1980s, ABC was a general-purpose language and programming environment. Guido van Rossum, the creator of Python, contributed to the development of the ABC language while working on a project known as Amoeba, a distributed operating system. In an interview, he shared his appreciation for ABC, recognizing its significant influence on his work in creating Python. Van Rossum wanted to create a language that retained ABC's strengths but addressed its shortcomings. Drawing from his experience with ABC, he designed Python, incorporating features like simple syntax, indentation-based code grouping instead of braces, and powerful data types such as dictionaries, lists, strings, and numbers. This blend of simplicity and functionality laid the foundation for Python's evolution into the versatile language it is today."

What is Machine Learning:-

Before delving into the various machine learning approaches, it is crucial to understand what machine learning entails and what it does not. Although it is frequently regarded as a subset of artificial intelligence, this perception can sometimes be inaccurate. While machine learning has its roots in AI research, it is more effectively described in data science as a methodology for developing models that interpret and represent data patterns.

At its core, machine learning is about constructing mathematical models that aid in understanding data. The "learning" aspect comes into play when these models have adjustable parameters that can be fine-tuned based on observed data. In this sense, the program can be thought of as "learning" from the data. Once these models are trained on historical data, they can be used to make predictions and draw insights from new, unseen data. The discussion about how this form of model-driven "learning" parallels human cognitive learning can be set aside. For now, the focus should be on grasping the context of a machine learning problem to apply these techniques effectively. Therefore, we will start with a broad overview of the various approaches we will examine.

Categories of Machine Learning:-

At the most fundamental level, machine learning can be categorized into several types.

Supervised learning is the most common type of machine learning. In this approach, the algorithm is trained on labeled data, meaning that the input data comes with corresponding correct output labels. The goal is to learn a mapping from inputs to outputs, so that the model can predict the output for unseen data.

In unsupervised learning, the algorithm deals with data that lacks labels. The objective is to uncover patterns, structures, or connections within the data, without any prior knowledge of the results.

Deep learning is a subset of machine learning, often considered its own category due to the use of neural networks with many layers (deep networks). It excels at handling large amounts of unstructured data, like images, audio, and text. Deep learning models automatically learn features from the raw data without the need for explicit feature engineering.

Need for Machine Learning

Humans are currently the most intelligent and advanced species on Earth due to their ability to think critically, analyze situations, and solve complex problems. In contrast, AI is still in its infancy and has not yet reached the level of human intelligence in many areas. This brings up an important question: why do we need machine learning? The key reason is that it empowers machines to analyze data, make informed decisions, and operate efficiently on a large scale".

Lately, organizations have been making substantial investments in cutting-edge technologies like AI, Machine Learning, and Deep Learning to gain valuable knowledge from data and solve a wide range of real-world issues. These technologies allow machines to make decisions derived from data, automating tasks that would otherwise require human input. Instead of relying solely on traditional programming logic, machine learning provides solutions for problems that cannot be easily encoded with rules. While human intelligence remains essential, the increasing need to address complex problems with efficiency and scale is why machine learning has become so crucial.

Challenges in Machines Learning:-

Machine learning (ML) faces several challenges that can affect its effectiveness and deployment. These challenges range from data-related issues like quality and availability, to technical concerns such as model interpretability and computational resource demands. Overcoming these hurdles is crucial for making ML systems more reliable, fair, and scalable in real-world applications

- Data Quality and Availability − Accurate models require high-quality, labeled data, which can be difficult and expensive to obtain. Poor or biased data can lead to unreliable results.
- Overfitting and Underfitting − Overfitting occurs when a model is too complex and fits noise, while underfitting happens when it's too simple. Both hinder the model's ability to generalize effectively.
- Model Interpretability − Many ML models, particularly deep learning, are complex and lack transparency. This makes it difficult to understand how they make decisions.
- Computational Resources − Advanced models need significant computational power for training, which can be a barrier for smaller organizations. Limited resources can delay or restrict model development.
- Bias and Fairness − ML models can unintentionally learn biases from training data, leading to unfair outcomes. Addressing bias is critical for ethical and reliable model deployment.
- Scalability − as data volumes increase, ensuring models scale efficiently without compromising performance is a major challenge. Large datasets require optimized infrastructure.

- Model Evaluation and Validation − Proper evaluation involves selecting the right metrics and ensuring the model generalizes well. Without this, models may perform poorly on unseen data.

Applications of Machine Learning:-

Machine Learning is the most rapidly growing technology and according to researchers we are in the golden year of AI and ML. It is used to solve many real-world complex problems which cannot be solved with traditional approach. Following are some real-world applications of ML

- Self-driving Cars
- Healthcare Diagnostics
- Speech Recognition
- Fraud Detection in Banking and Finance
- Recommendation Systems (e.g., Netflix, Amazon)
- Speech Recognition (e.g., Siri, Google Assistant)
- Email Filtering (e.g., spam detection)
- Predictive Maintenance in Manufacturing
- Face Recognition (e.g., security systems, social media)
- Customer Service Chatbots (e.g., in banking or retail)
- Credit Scoring and Risk Assessment
- Social Media Sentiment Analysis
- Supply Chain Optimization (e.g., inventory management)

Based on your melanoma detection project, the following deep learning terminologies are directly relevant

- **Model –** The convolutional neural network (CNN) created to classify skin diseases, trained on image data from the ISIC dataset.
- **Feature –** In this context, the individual properties of the images, such as pixel values, that are used as input to the model.

- **Target –** The disease category or class (e.g., melanoma, benign keratosis) that the model is trying to predict.
- **Training –** The process of feeding the images (features) and their corresponding disease categories (labels) into the model to learn the classification task.
- **Prediction –** Once trained, the model predicts the disease category for a new image based on the learned features.

SCREEN SHOTS

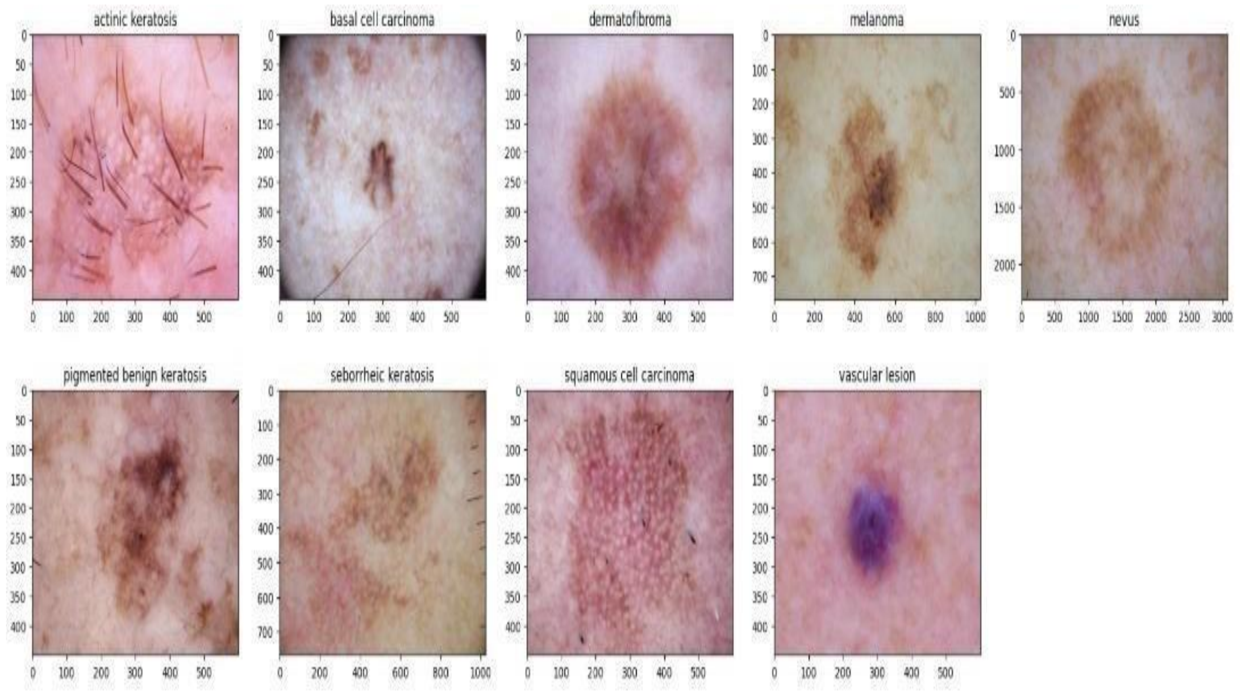
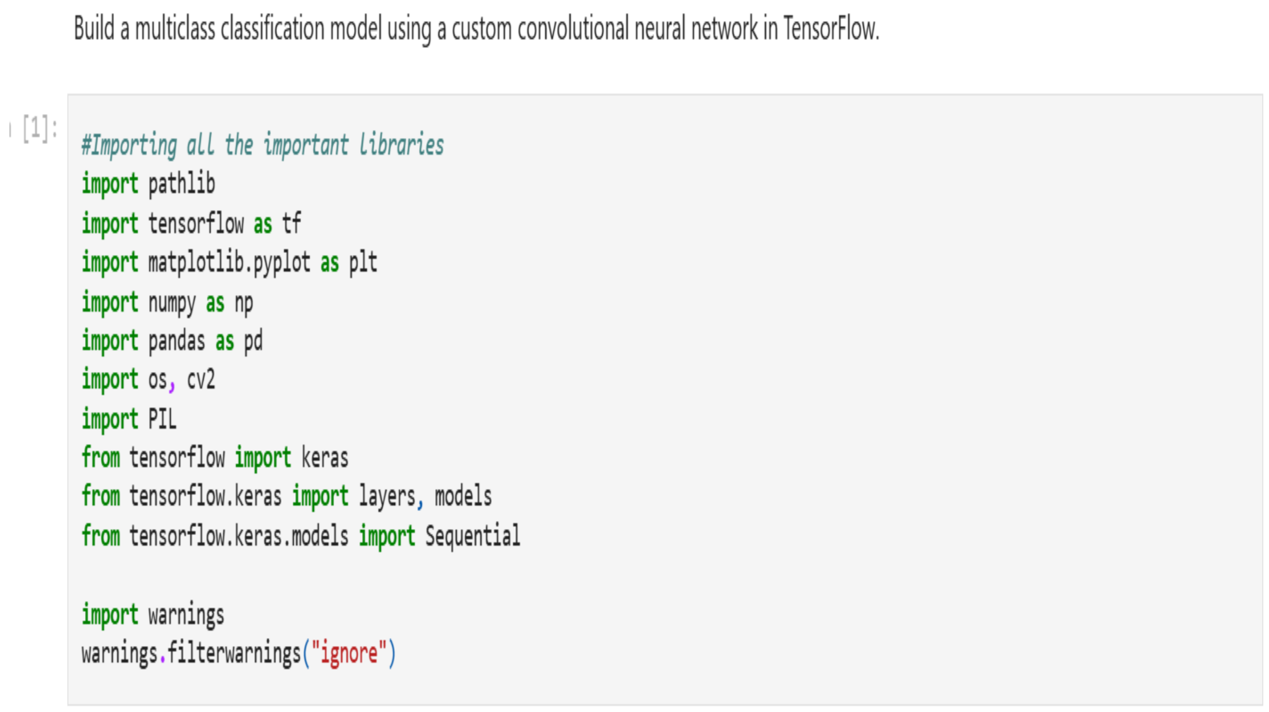
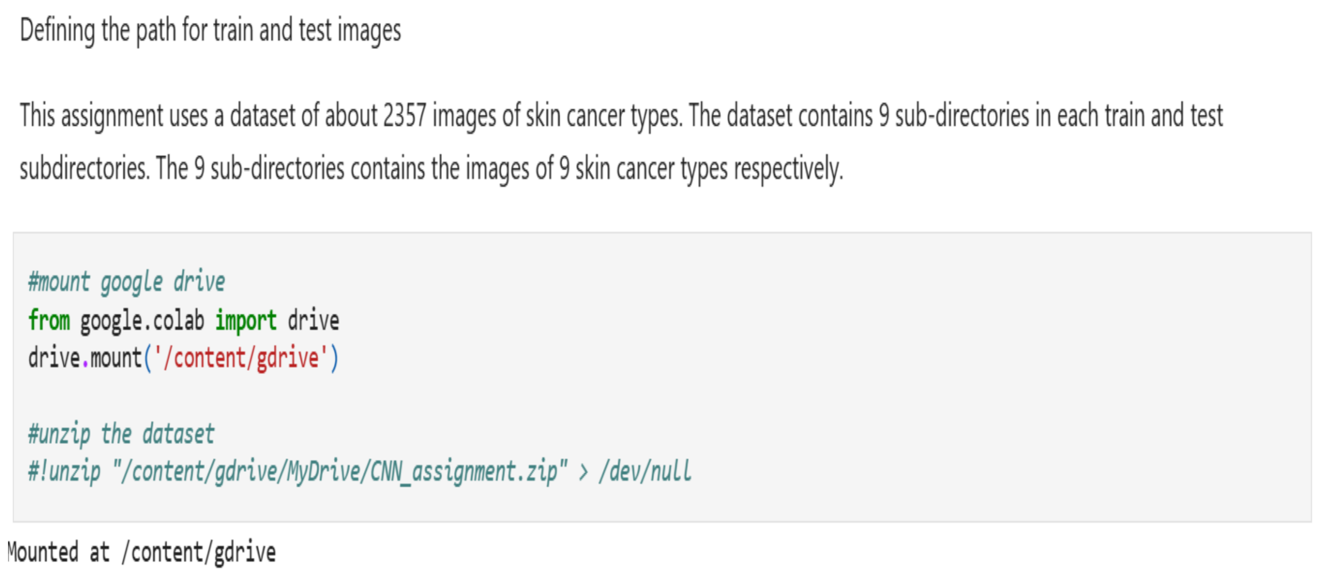


Figure 6 : Screen shot of DataSet

Data Reading / Data Understanding


Build a multiclass classification model using a custom convolutional neural network in TensorFlow.

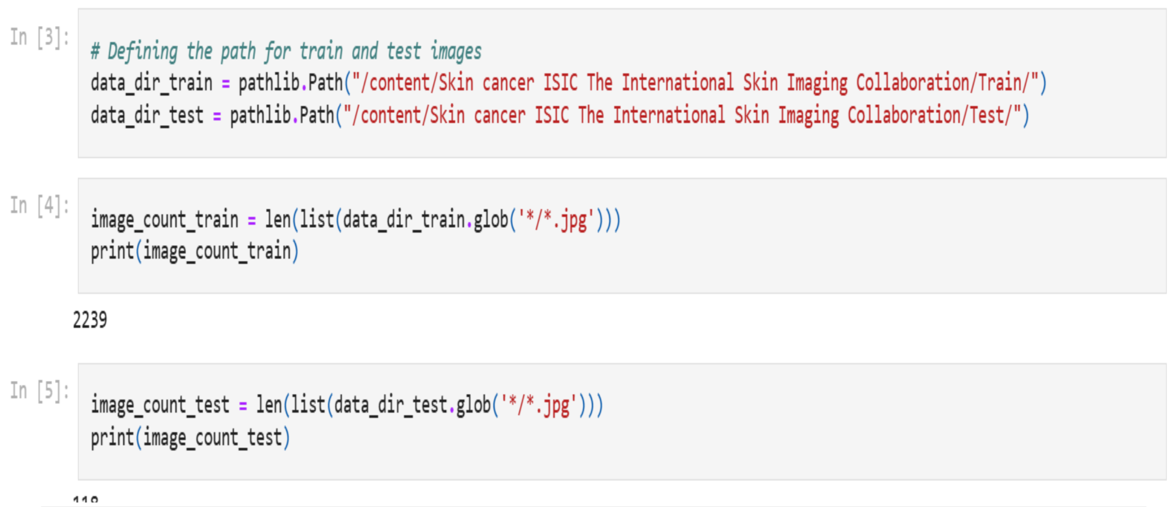
[1]:
```
#Importing all the important libraries
import pathlib
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os, cv2
import PIL
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")
```


Defining the path for train and test images

This assignment uses a dataset of about 2357 images of skin cancer types. The dataset contains 9 sub-directories in each train and test subdirectories. The 9 sub-directories contains the images of 9 skin cancer types respectively.

```
#mount google drive
from google.colab import drive
drive.mount('/content/gdrive')

#unzip the dataset
#!unzip "/content/gdrive/MyDrive/CNN_assignment.zip" > /dev/null
```

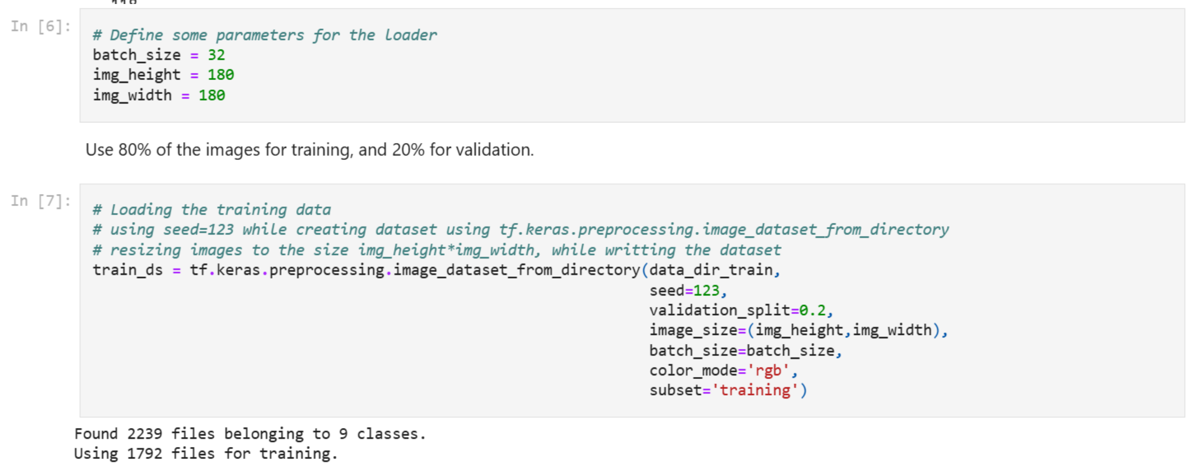
```
Mounted at /content/gdrive
```



```
In [3]: # Defining the path for train and test images
        data_dir_train = pathlib.Path("/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/")
        data_dir_test = pathlib.Path("/content/Skin cancer ISIC The International Skin Imaging Collaboration/Test/")
```

```
In [4]: image_count_train = len(list(data_dir_train.glob('*/*.jpg')))
        print(image_count_train)

        2239
```

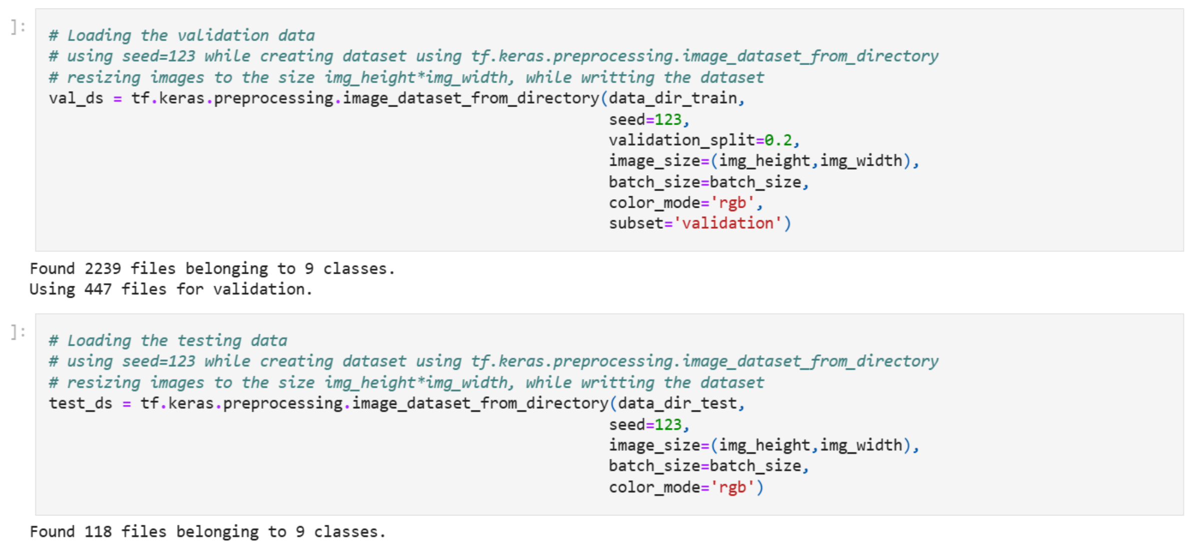
```
In [5]: image_count_test = len(list(data_dir_test.glob('*/*.jpg')))
        print(image_count_test)

        118
```



```
In [6]: # Define some parameters for the loader
        batch_size = 32
        img_height = 180
        img_width = 180
```

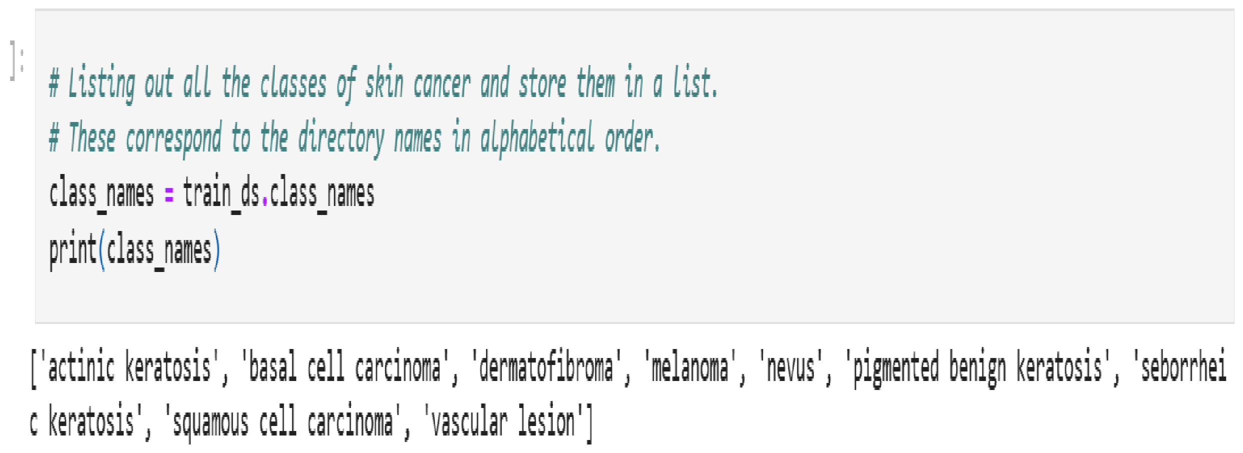
Use 80% of the images for training, and 20% for validation.

```
In [7]: # Loading the training data
        # using seed=123 while creating dataset using tf.keras.preprocessing.image_dataset_from_directory
        # resizing images to the size img_height*img_width, while writting the dataset
        train_ds = tf.keras.preprocessing.image_dataset_from_directory(data_dir_train,
                                                                       seed=123,
                                                                       validation_split=0.2,
                                                                       image_size=(img_height,img_width),
                                                                       batch_size=batch_size,
                                                                       color_mode='rgb',
                                                                       subset='training')

        Found 2239 files belonging to 9 classes.
        Using 1792 files for training.
```



```
]: # Loading the validation data
   # using seed=123 while creating dataset using tf.keras.preprocessing.image_dataset_from_directory
   # resizing images to the size img_height*img_width, while writting the dataset
   val_ds = tf.keras.preprocessing.image_dataset_from_directory(data_dir_train,
                                                                seed=123,
                                                                validation_split=0.2,
                                                                image_size=(img_height,img_width),
                                                                batch_size=batch_size,
                                                                color_mode='rgb',
                                                                subset='validation')

   Found 2239 files belonging to 9 classes.
   Using 447 files for validation.
```

```
]: # Loading the testing data
   # using seed=123 while creating dataset using tf.keras.preprocessing.image_dataset_from_directory
   # resizing images to the size img_height*img_width, while writting the dataset
   test_ds = tf.keras.preprocessing.image_dataset_from_directory(data_dir_test,
                                                                 seed=123,
                                                                 image_size=(img_height,img_width),
                                                                 batch_size=batch_size,
                                                                 color_mode='rgb')

   Found 118 files belonging to 9 classes.
```



```
# Listing out all the classes of skin cancer and store them in a list.
# These correspond to the directory names in alphabetical order.
class_names = train_ds.class_names
print(class_names)
```

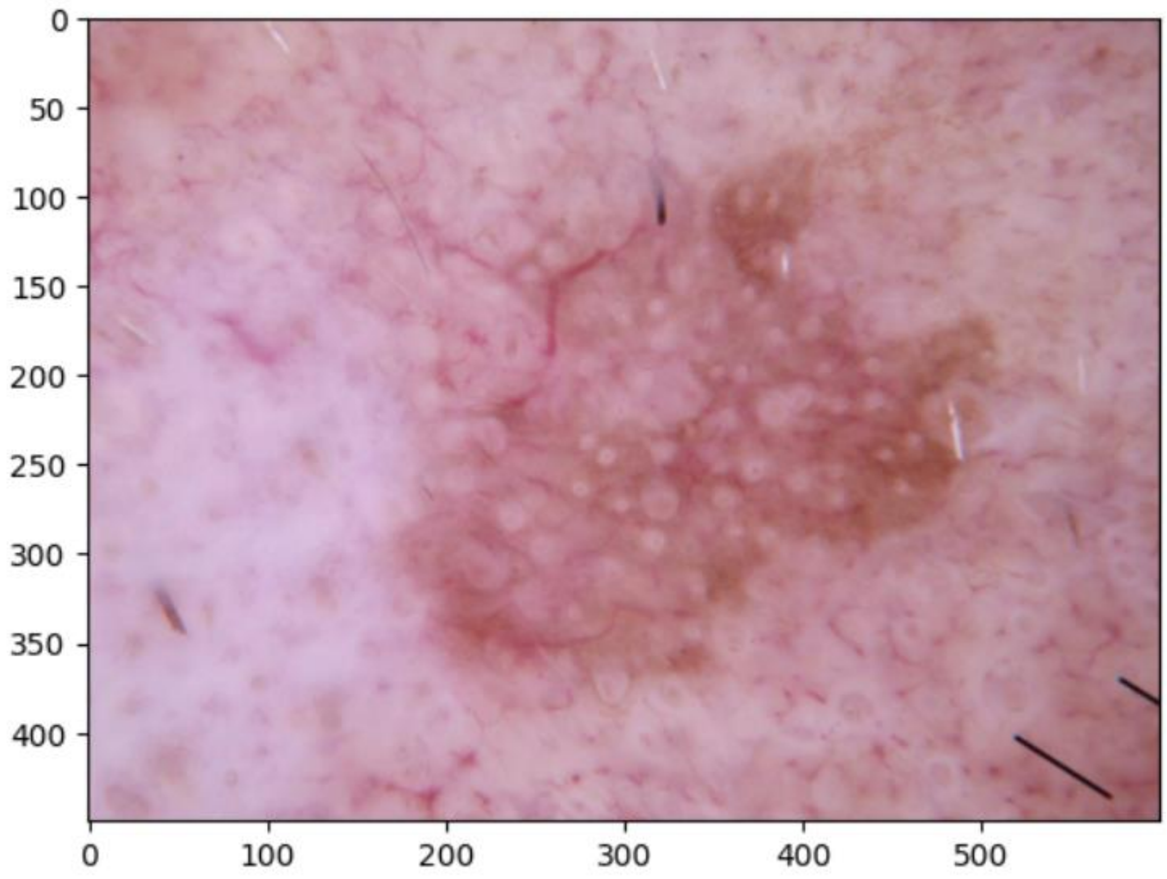
```
['actinic keratosis', 'basal cell carcinoma', 'dermatofibroma', 'melanoma', 'nevus', 'pigmented benign keratosis', 'seborrhei
c keratosis', 'squamous cell carcinoma', 'vascular lesion']
```

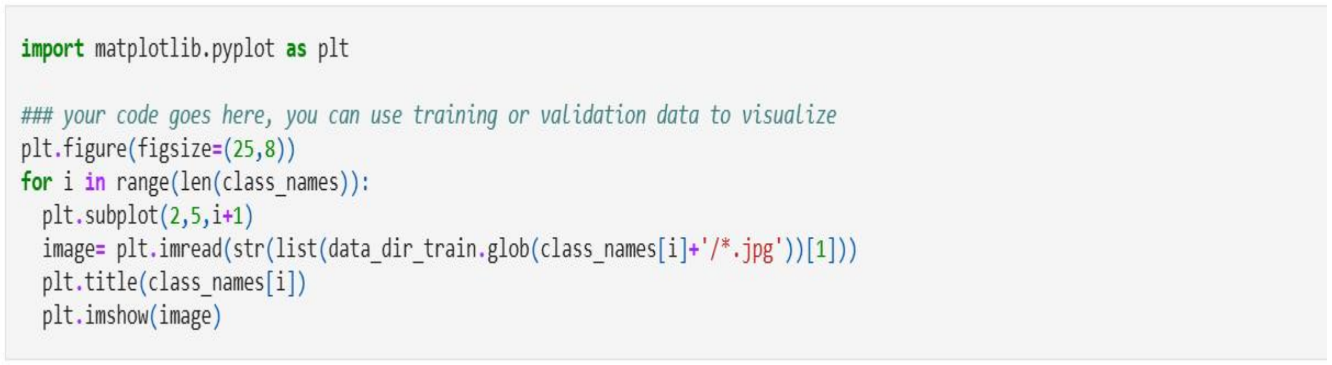
Figure 7 : Data Reading or Understanding

Data Visualization

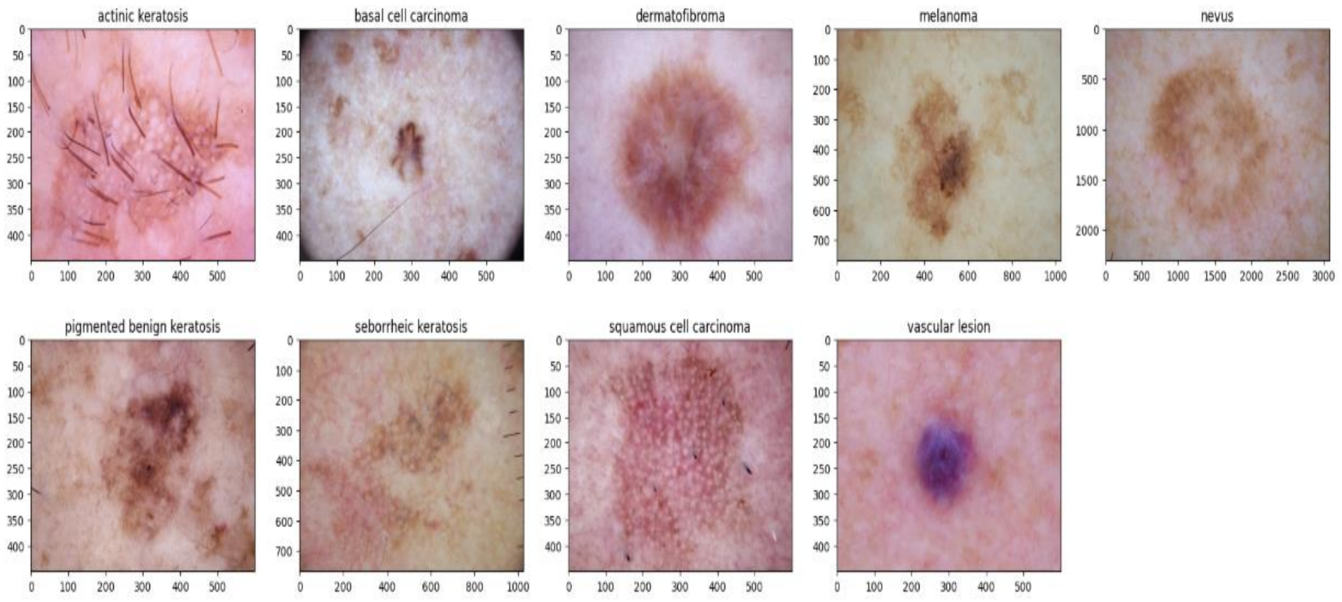
```
image = plt.imread((list(data_dir_train.glob(class_names[0]+'/*.jpg'))[0]))
plt.imshow(image)
```

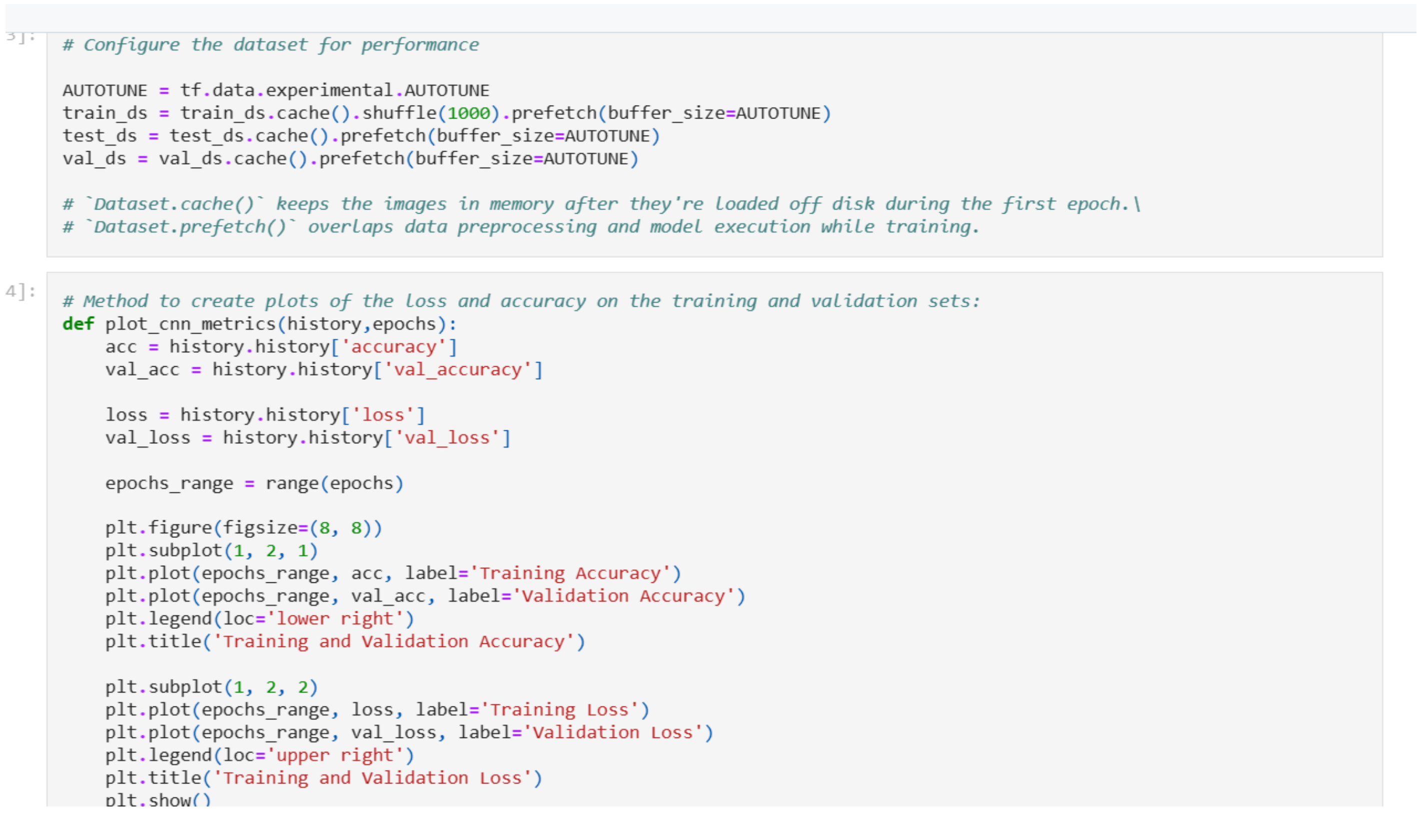
```
<matplotlib.image.AxesImage at 0x799b13f58730>
```





```python
import matplotlib.pyplot as plt

### your code goes here, you can use training or validation data to visualize
plt.figure(figsize=(25,8))
for i in range(len(class_names)):
  plt.subplot(2,5,i+1)
  image= plt.imread(str(list(data_dir_train.glob(class_names[i]+'/*.jpg'))[1]))
  plt.title(class_names[i])
  plt.imshow(image)
```



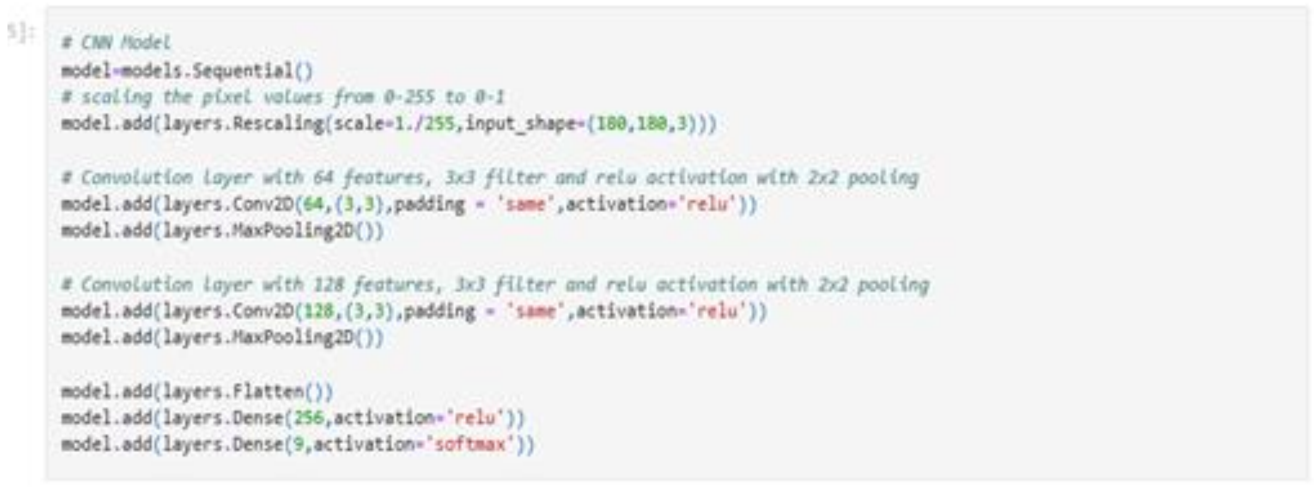


```python
# Configure the dataset for performance

AUTOTUNE = tf.data.experimental.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# `Dataset.cache()` keeps the images in memory after they're loaded off disk during the first epoch.\
# `Dataset.prefetch()` overlaps data preprocessing and model execution while training.
```

```python
# Method to create plots of the loss and accuracy on the training and validation sets:
def plot_cnn_metrics(history,epochs):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(epochs)

    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()
```

Figure 8 : Data Visualization

Model Building & Training


```
# CNN Model
model=models.Sequential()
# scaling the pixel values from 0-255 to 0-1
model.add(layers.Rescaling(scale=1./255,input_shape=(180,180,3)))

# Convolution layer with 64 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(64,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

# Convolution layer with 128 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(128,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(9,activation='softmax'))
```

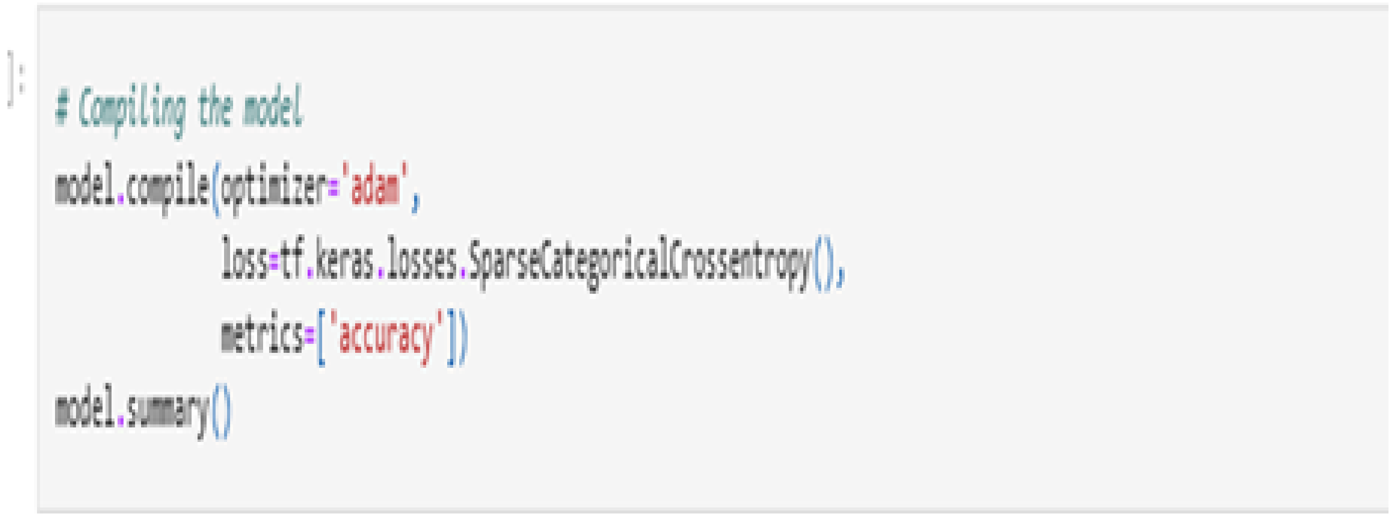
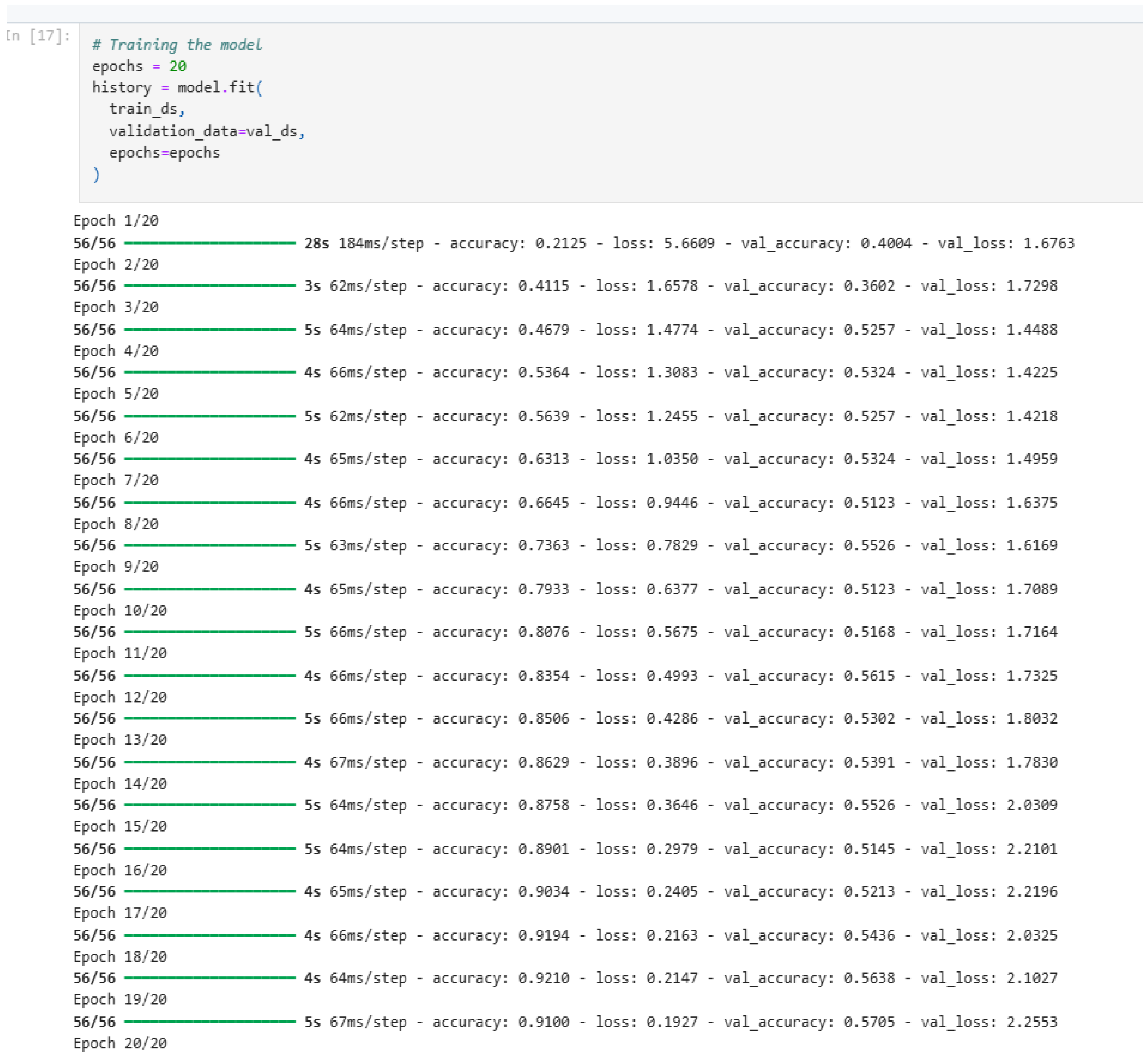

```
# Compiling the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
model.summary()
```

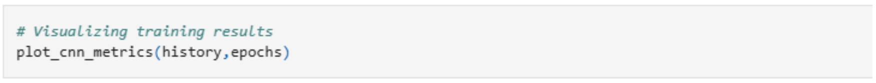
Figure 9 : Rescaling Shuffling Normalization



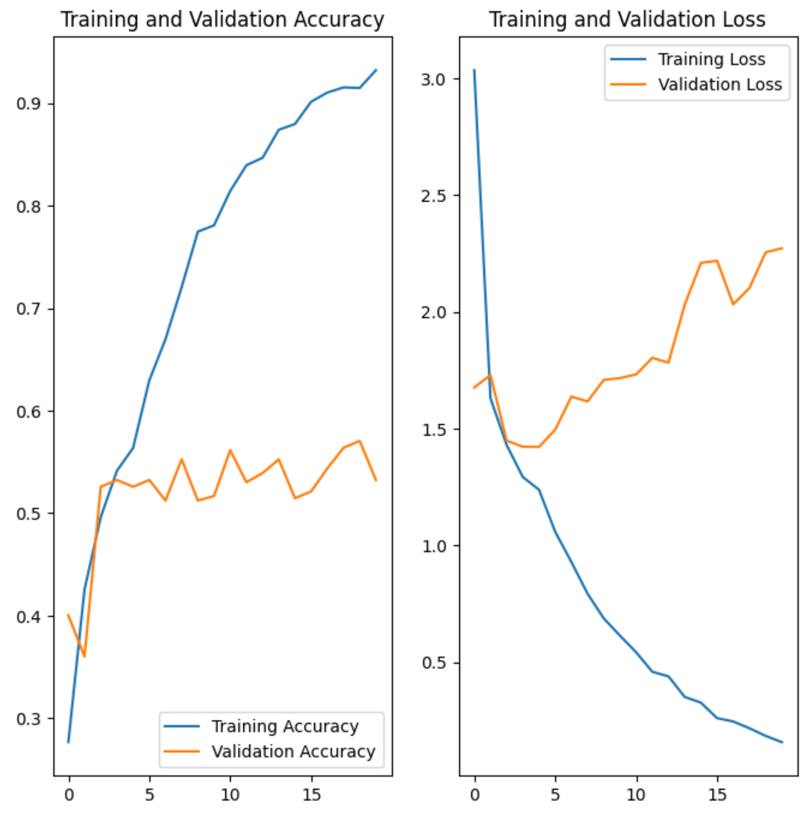
```
In [17]:
# Training the model
epochs = 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

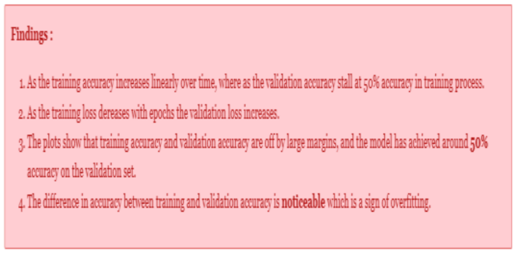
```
Epoch 1/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 28s 184ms/step - accuracy: 0.2125 - loss: 5.6609 - val_accuracy: 0.4004 - val_loss: 1.6763
Epoch 2/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step - accuracy: 0.4115 - loss: 1.6578 - val_accuracy: 0.3602 - val_loss: 1.7298
Epoch 3/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 64ms/step - accuracy: 0.4679 - loss: 1.4774 - val_accuracy: 0.5257 - val_loss: 1.4488
Epoch 4/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.5364 - loss: 1.3083 - val_accuracy: 0.5324 - val_loss: 1.4225
Epoch 5/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 62ms/step - accuracy: 0.5639 - loss: 1.2455 - val_accuracy: 0.5257 - val_loss: 1.4218
Epoch 6/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.6313 - loss: 1.0350 - val_accuracy: 0.5324 - val_loss: 1.4959
Epoch 7/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.6645 - loss: 0.9446 - val_accuracy: 0.5123 - val_loss: 1.6375
Epoch 8/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 63ms/step - accuracy: 0.7363 - loss: 0.7829 - val_accuracy: 0.5526 - val_loss: 1.6169
Epoch 9/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.7933 - loss: 0.6377 - val_accuracy: 0.5123 - val_loss: 1.7089
Epoch 10/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 66ms/step - accuracy: 0.8076 - loss: 0.5675 - val_accuracy: 0.5168 - val_loss: 1.7164
Epoch 11/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.8354 - loss: 0.4993 - val_accuracy: 0.5615 - val_loss: 1.7325
Epoch 12/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 66ms/step - accuracy: 0.8506 - loss: 0.4286 - val_accuracy: 0.5302 - val_loss: 1.8032
Epoch 13/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.8629 - loss: 0.3896 - val_accuracy: 0.5391 - val_loss: 1.7830
Epoch 14/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 64ms/step - accuracy: 0.8758 - loss: 0.3646 - val_accuracy: 0.5526 - val_loss: 2.0309
Epoch 15/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 64ms/step - accuracy: 0.8901 - loss: 0.2979 - val_accuracy: 0.5145 - val_loss: 2.2101
Epoch 16/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.9034 - loss: 0.2405 - val_accuracy: 0.5213 - val_loss: 2.2196
Epoch 17/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.9194 - loss: 0.2163 - val_accuracy: 0.5436 - val_loss: 2.0325
Epoch 18/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - accuracy: 0.9210 - loss: 0.2147 - val_accuracy: 0.5638 - val_loss: 2.1027
Epoch 19/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 67ms/step - accuracy: 0.9100 - loss: 0.1927 - val_accuracy: 0.5705 - val_loss: 2.2553
Epoch 20/20
```

Figure 10 : Model Training



```
# Visualizing training results
plot_cnn_metrics(history,epochs)
```





Findings :

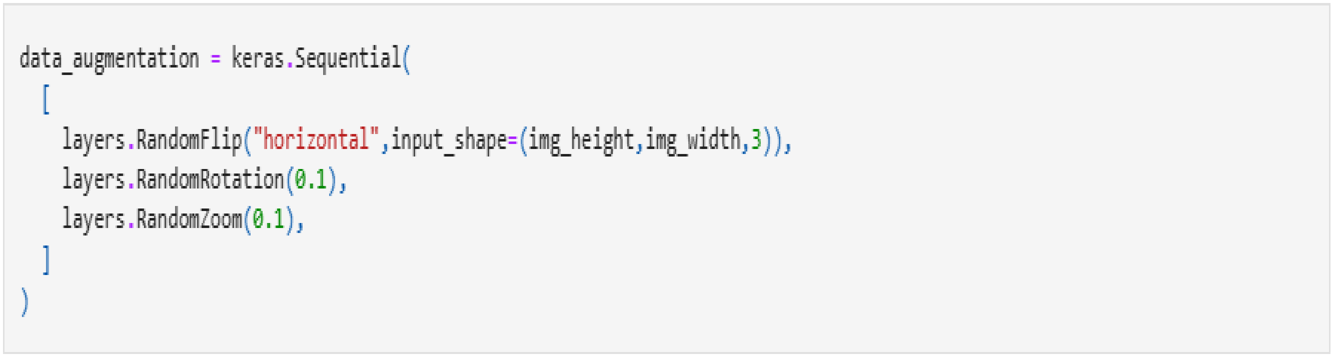
1. As the training accuracy increases linearly over time, where as the validation accuracy stall at 50% accuracy in training process.
2. As the training loss dereases with epochs the validation loss increases.
3. The plots show that training accuracy and validation accuracy are off by large margins, and the model has achieved around **50%** accuracy on the validation set.
4. The difference in accuracy between training and validation accuracy is **noticeable** which is a sign of overfitting.

Figure 11 : Training, Validation and Findings

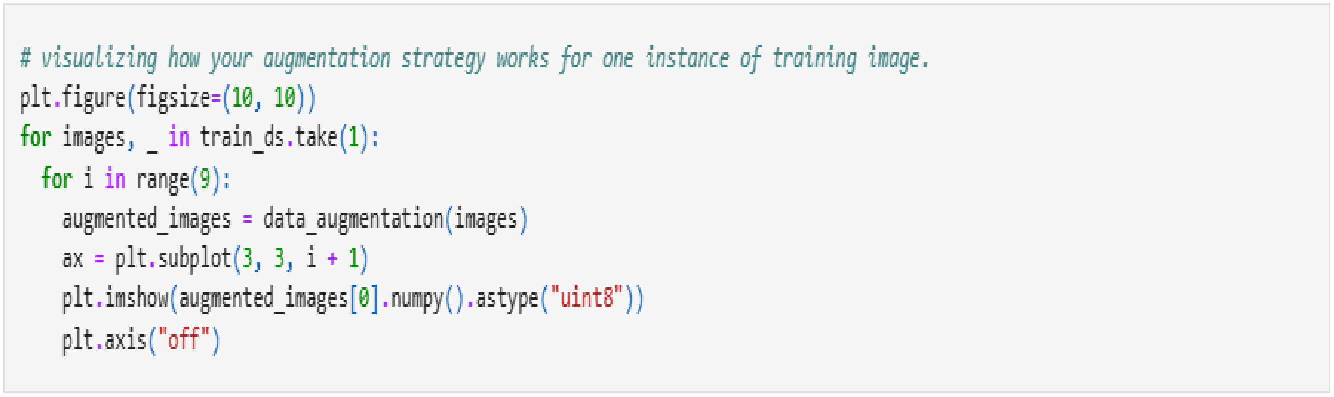
Data Augmentation

Choosing an appropriate data augmentation strategy to resolve underfitting/overfitting

Overfitting generally occurs when there are a small number of training examples. Data augmentation takes the approach of generating additional training data from your existing examples by augmenting them using random transformations that yield believable-looking images. This helps expose the model to more aspects of the data and generalize better.



```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",input_shape=(img_height,img_width,3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```



```
# visualizing how your augmentation strategy works for one instance of training image.
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

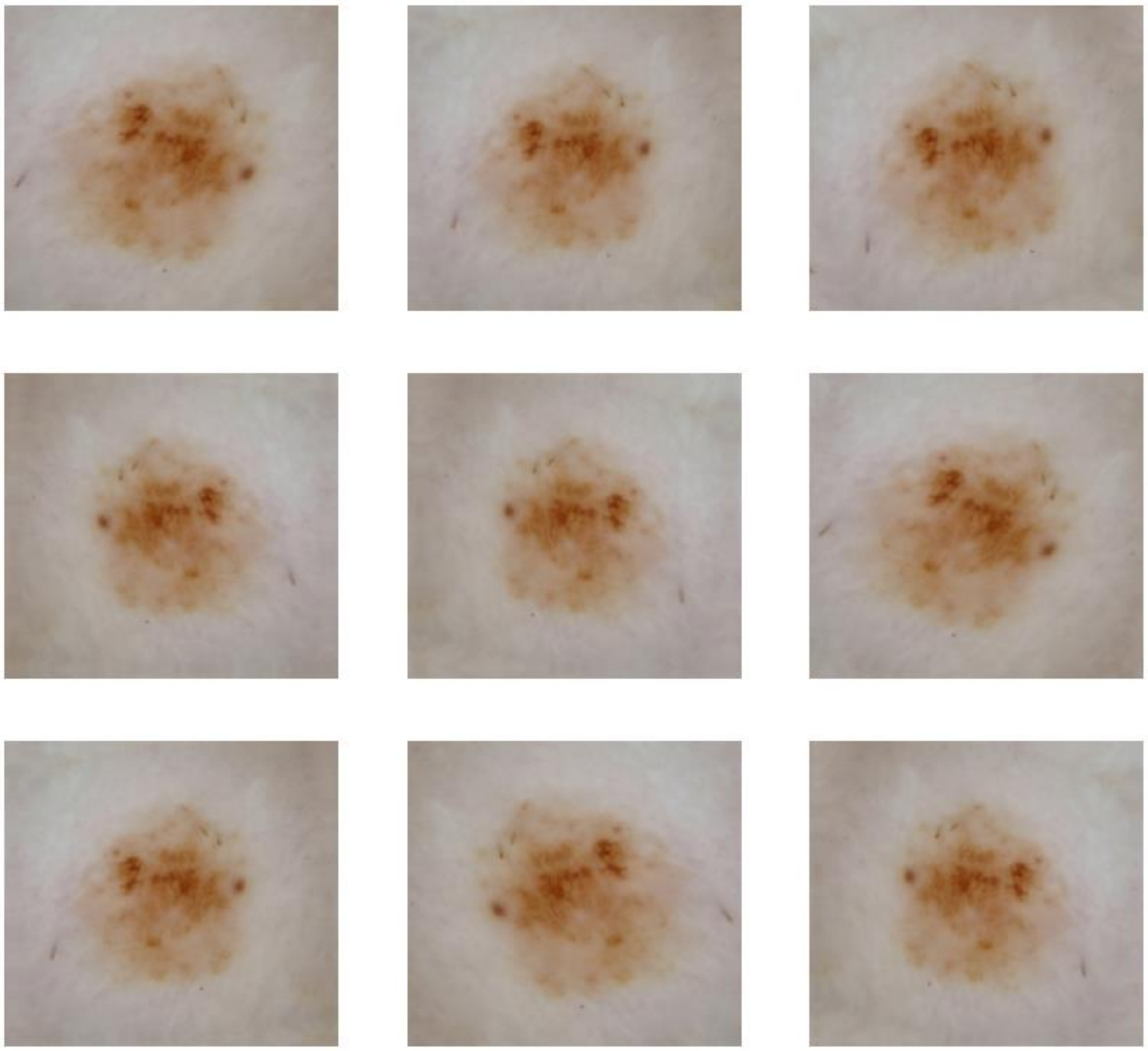
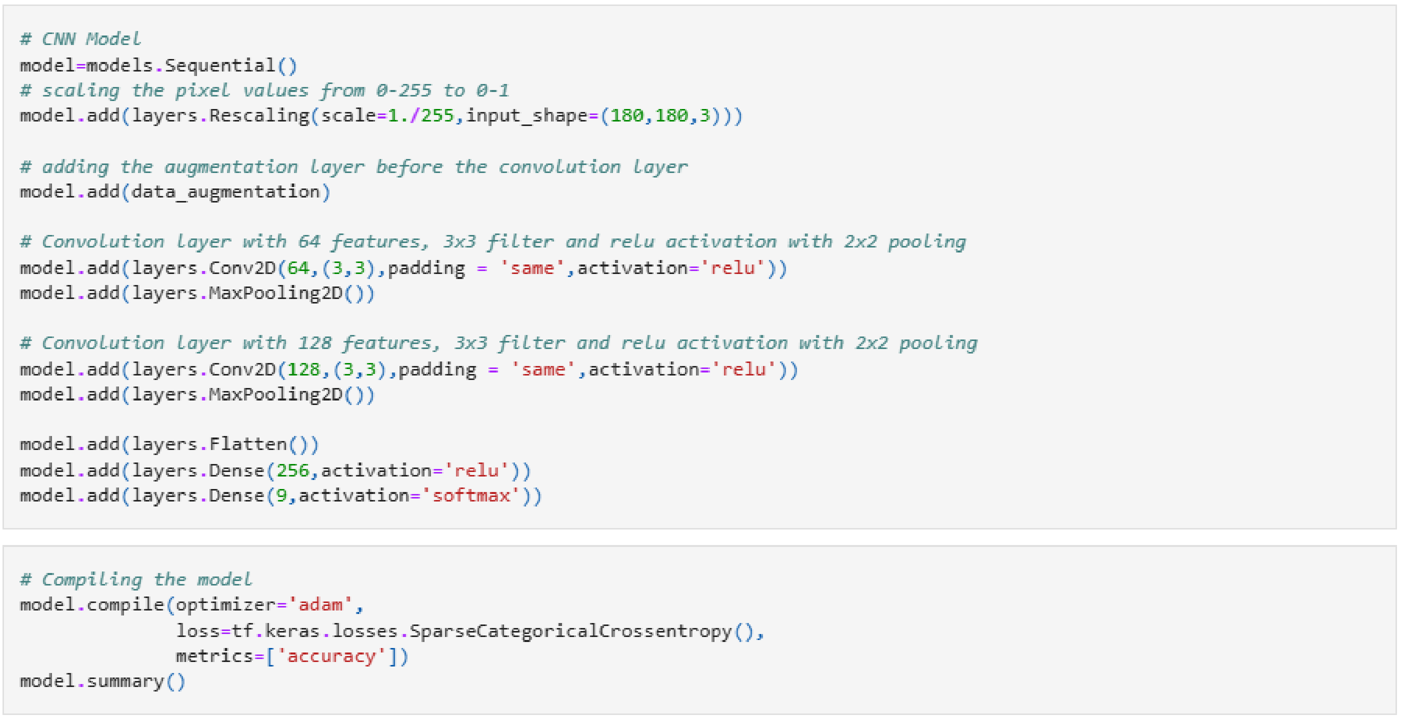


Figure 12 : Resolving Overfitting & Underfitting

Model Building & training on the augmented data

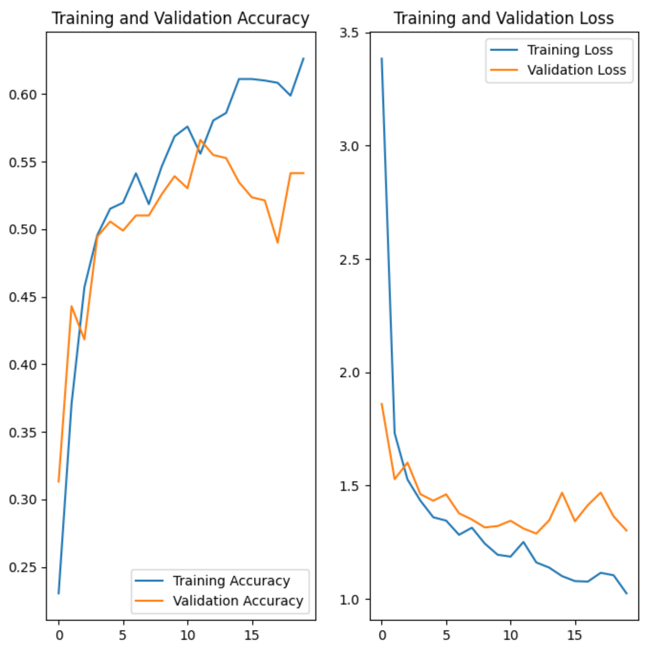


```
# CNN Model
model=models.Sequential()
# scaling the pixel values from 0-255 to 0-1
model.add(layers.Rescaling(scale=1./255,input_shape=(180,180,3)))

# adding the augmentation layer before the convolution layer
model.add(data_augmentation)

# Convolution layer with 64 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(64,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

# Convolution layer with 128 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(128,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(9,activation='softmax'))
```

```
# Compiling the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
model.summary()
```

Figure 13 : Training on Augmented data

```
# Visualizing training results
plot_cnn_metrics(history,epochs)
```



Findings :

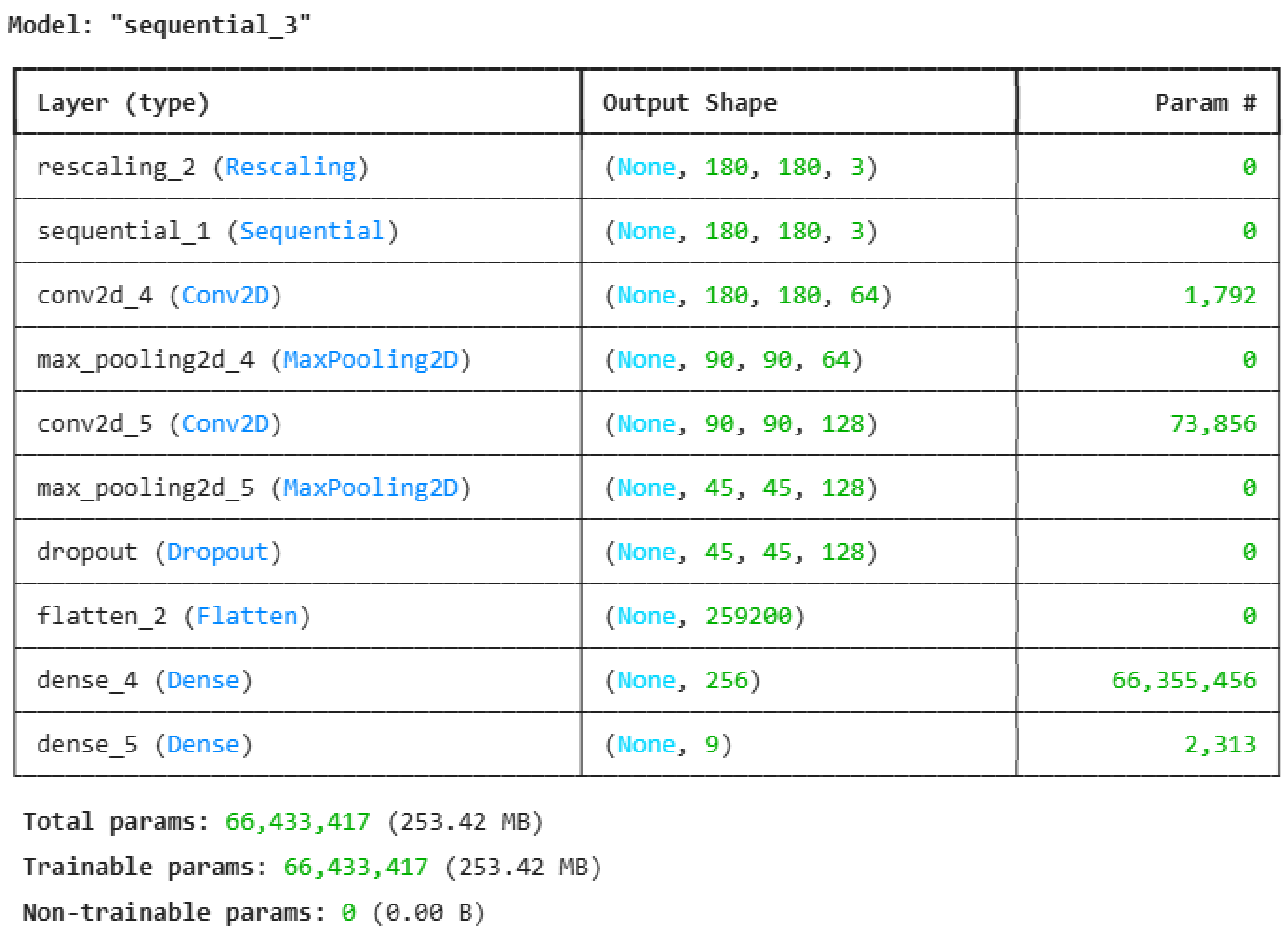
1. As the training accuracy increases linearly over time, where as the validation accuracy increases and stall at 55% accuracy in training process.
2. As the training loss decreases with epochs the validation loss decreases and stalls.
3. The plots show that gap between training accuracy and validation accuracy have decreased from previous model, and it has achieved around **55%** accuracy on the validation set.
4. The difference in accuracy between training and validation accuracy is still **slightly noticeable** which is a sign of overfitting.

Figure 14 : Validation Accuracy, Findings with Augmented data

Model Building & training on the augmented data with dropout

```
# CNN Model
model=models.Sequential()
# scaling the pixel values from 0-255 to 0-1
model.add(layers.Rescaling(scale=1./255,input_shape=(180,180,3)))
model.add(data_augmentation)

# Convolution layer with 64 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(64,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

# Convolution layer with 128 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(128,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())
#adding a 20% dropout after the convolution layers
model.add(layers.Dropout(0.2))

model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(9,activation='softmax'))
```

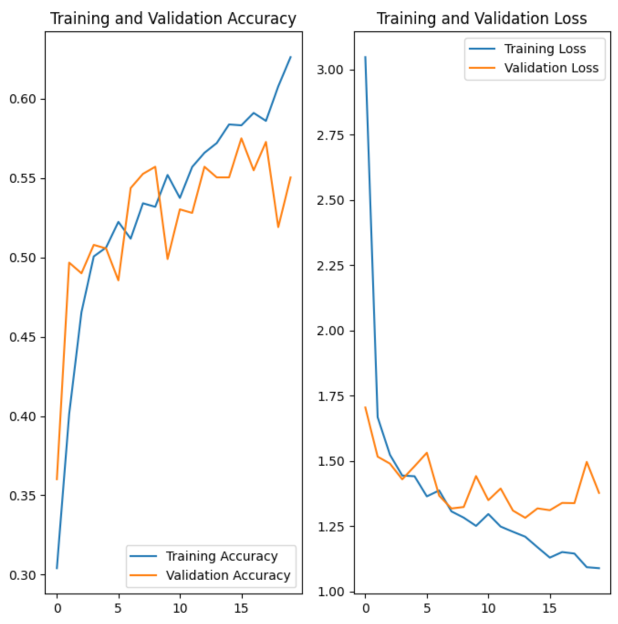
```
# Compiling the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
model.summary()
```



```
Model: "sequential_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ rescaling_2 (Rescaling)              │ (None, 180, 180, 3)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ sequential_1 (Sequential)            │ (None, 180, 180, 3)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_4 (Conv2D)                    │ (None, 180, 180, 64)        │           1,792 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_4 (MaxPooling2D)       │ (None, 90, 90, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_5 (Conv2D)                    │ (None, 90, 90, 128)         │          73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_5 (MaxPooling2D)       │ (None, 45, 45, 128)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 45, 45, 128)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_2 (Flatten)                  │ (None, 259200)              │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_4 (Dense)                      │ (None, 256)                 │      66,355,456 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_5 (Dense)                      │ (None, 9)                   │           2,313 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

 Total params: 66,433,417 (253.42 MB)

 Trainable params: 66,433,417 (253.42 MB)

 Non-trainable params: 0 (0.00 B)
```

Figure 15 : Training on Augmented data with dropout

```
# Visualizing training results
plot_cnn_metrics(history,epochs)
```



Findings :

1. As the training accuracy increases linearly over time, where as the validation accuracy increases and stall at 55% accuracy in training process.
2. **As the training loss decreases with epochs the validation loss decreases**
3. The plots show that gap between training accuracy and validation accuracy have decreased from previous model, and it has achieved around **55%** accuracy on the validation set.
4. The difference in accuracy between training and validation accuracy is **very less**

Figure 16 : Reduced Overfitting compared to earlier

Class distribution

Examining the current class distribution in the training dataset

Datasets can have class imbalance, one class can have proportionately higher number of samples compared to the others.

Class imbalance can have a detrimental effect on the final model quality. Hence as a sanity check it becomes important to check what is the distribution of classes in the data.

```
for i in range(len(class_names)):
  print(class_names[i],' - ',len(list(data_dir_train.glob(class_names[i]+'/*.jpg'))))
```

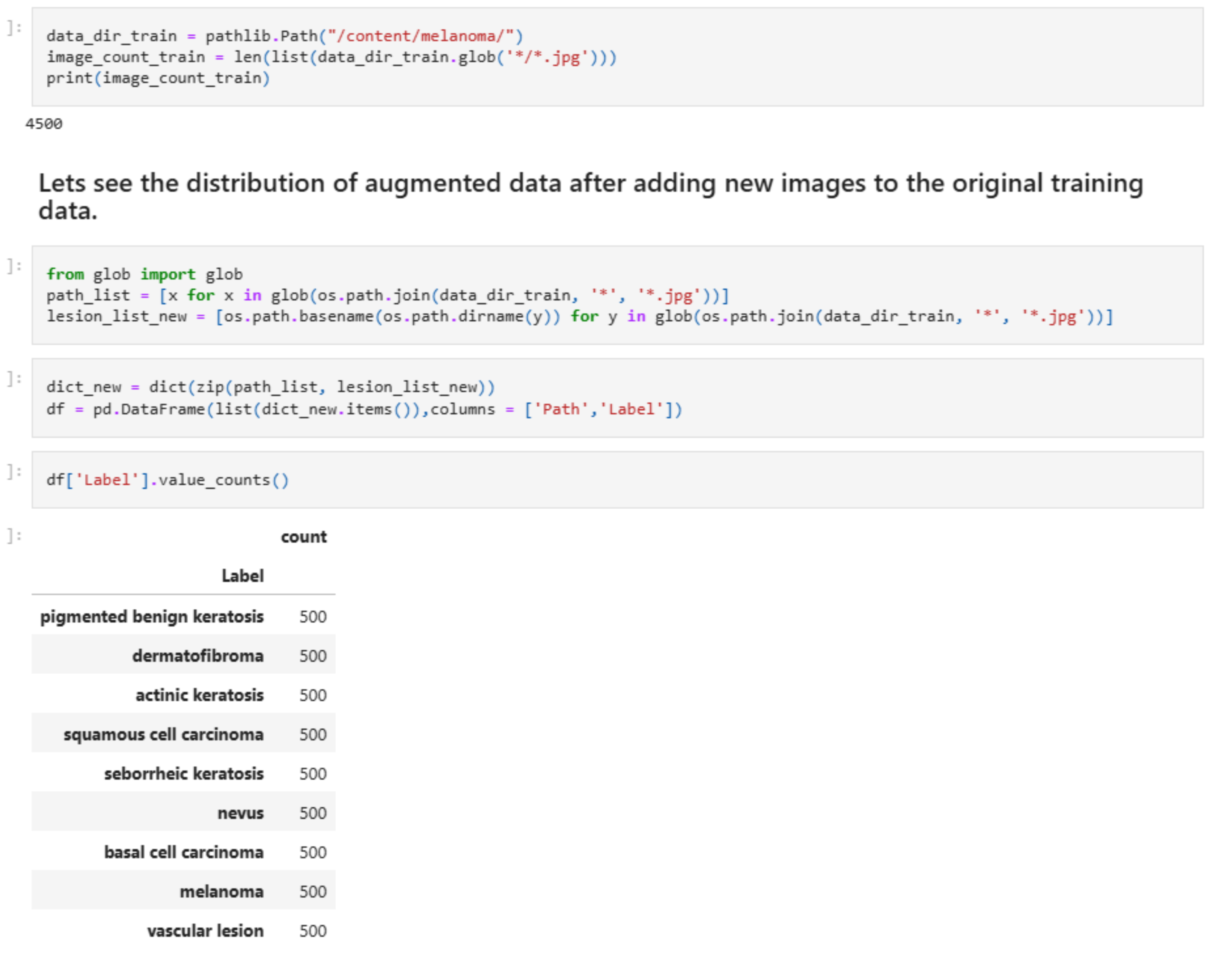
```
actinic keratosis  -  114
basal cell carcinoma  -  376
dermatofibroma  -  95
melanoma  -  438
nevus  -  357
pigmented benign keratosis  -  462
seborrheic keratosis  -  77
squamous cell carcinoma  -  181
vascular lesion  -  139
```

Findings :

1. Which class has the least number of samples?
   seborrheic keratosis with 77 samples
2. Which classes dominate the data in terms proportionate number of samples?
   pigmented benign keratosis with 462 samples

Figure 17 : Class Distribution

Handling class imbalances



```
]:  data_dir_train = pathlib.Path("/content/melanoma/")
    image_count_train = len(list(data_dir_train.glob('*/*.jpg')))
    print(image_count_train)

    4500
```

Lets see the distribution of augmented data after adding new images to the original training data.

```
]:  from glob import glob
    path_list = [x for x in glob(os.path.join(data_dir_train, '*', '*.jpg'))]
    lesion_list_new = [os.path.basename(os.path.dirname(y)) for y in glob(os.path.join(data_dir_train, '*', '*.jpg'))]
```

```
]:  dict_new = dict(zip(path_list, lesion_list_new))
    df = pd.DataFrame(list(dict_new.items()),columns = ['Path','Label'])
```

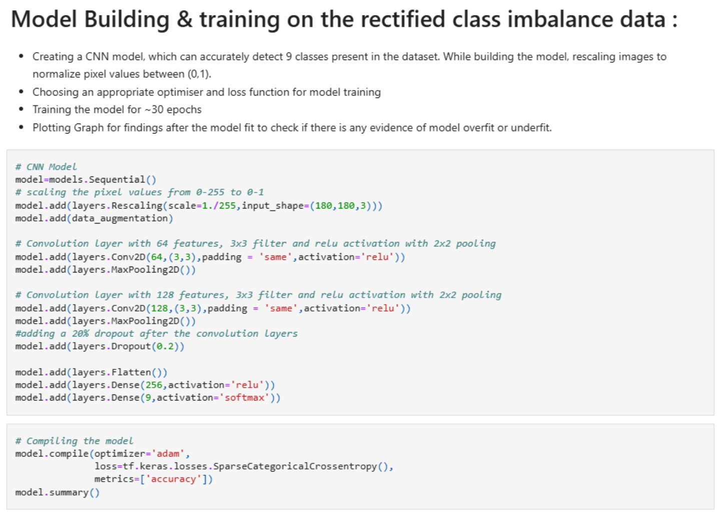
```
]:  df['Label'].value_counts()
```

]:

	count
Label	
pigmented benign keratosis	500
dermatofibroma	500
actinic keratosis	500
squamous cell carcinoma	500
seborrheic keratosis	500
nevus	500
basal cell carcinoma	500
melanoma	500
vascular lesion	500

Figure 18 : Class Imbalances

Final Model Training

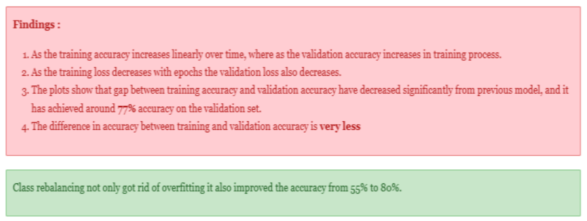


Model Building & training on the rectified class imbalance data :

- Creating a CNN model, which can accurately detect 9 classes present in the dataset. While building the model, rescaling images to normalize pixel values between (0,1).
- Choosing an appropriate optimiser and loss function for model training
- Training the model for ~30 epochs
- Plotting Graph for findings after the model fit to check if there is any evidence of model overfit or underfit.

```
# CNN Model
model=models.Sequential()
# scaling the pixel values from 0-255 to 0-1
model.add(layers.Rescaling(scale=1./255,input_shape=(180,180,3)))
model.add(data_augmentation)

# Convolution layer with 64 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(64,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

# Convolution layer with 128 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(128,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())
#adding a 20% dropout after the convolution layers
model.add(layers.Dropout(0.2))

model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(9,activation='softmax'))
```

```
# Compiling the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
model.summary()
```



Findings :

1. As the training accuracy increases linearly over time, where as the validation accuracy increases in training process.
2. As the training loss decreases with epochs the validation loss also decreases.
3. The plots show that gap between training accuracy and validation accuracy have decreased significantly from previous model, and it has achieved around 77% accuracy on the validation set.
4. The difference in accuracy between training and validation accuracy is **very less**

Class rebalancing not only got rid of overfitting it also improved the accuracy from 55% to 80%.

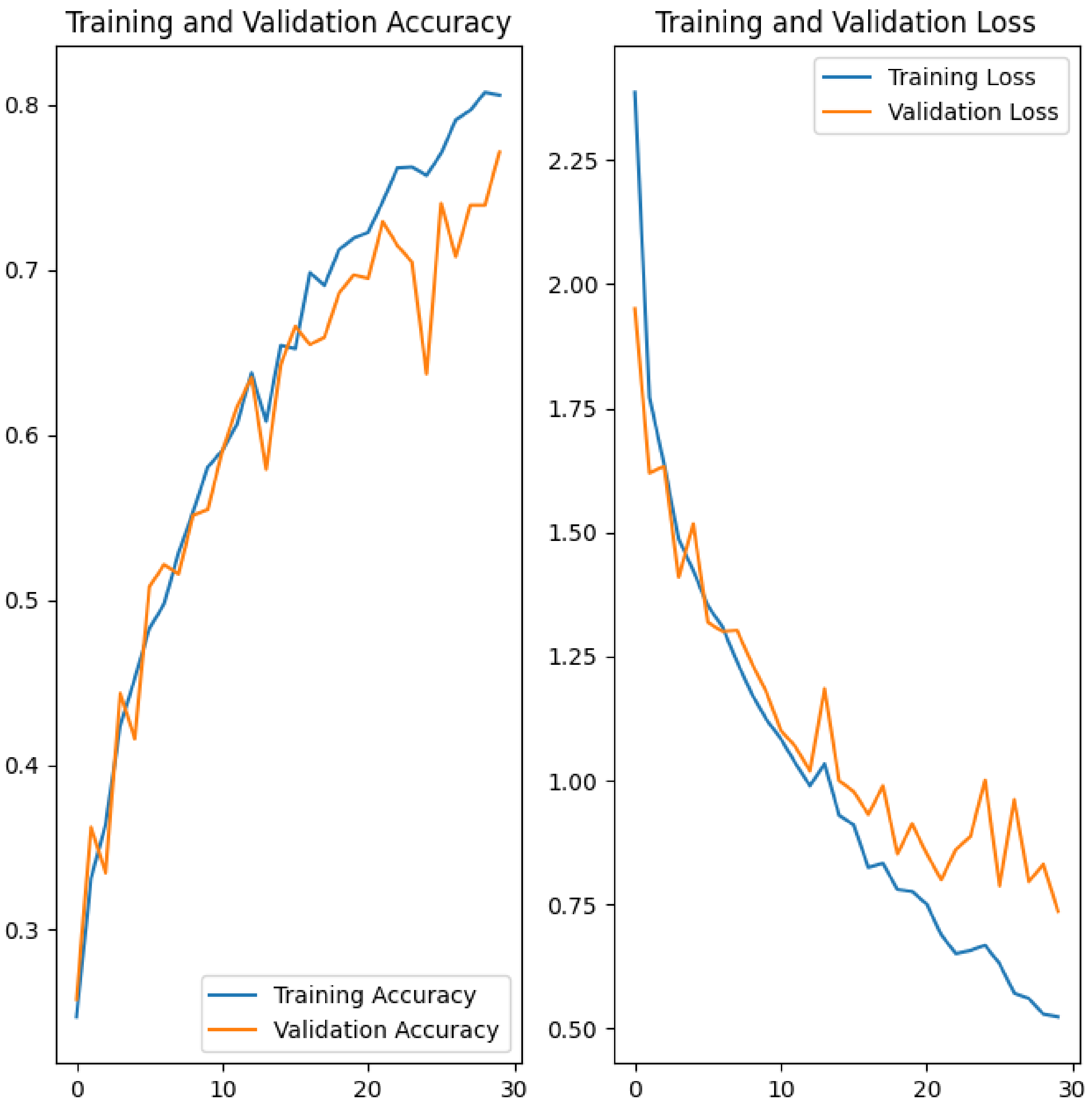
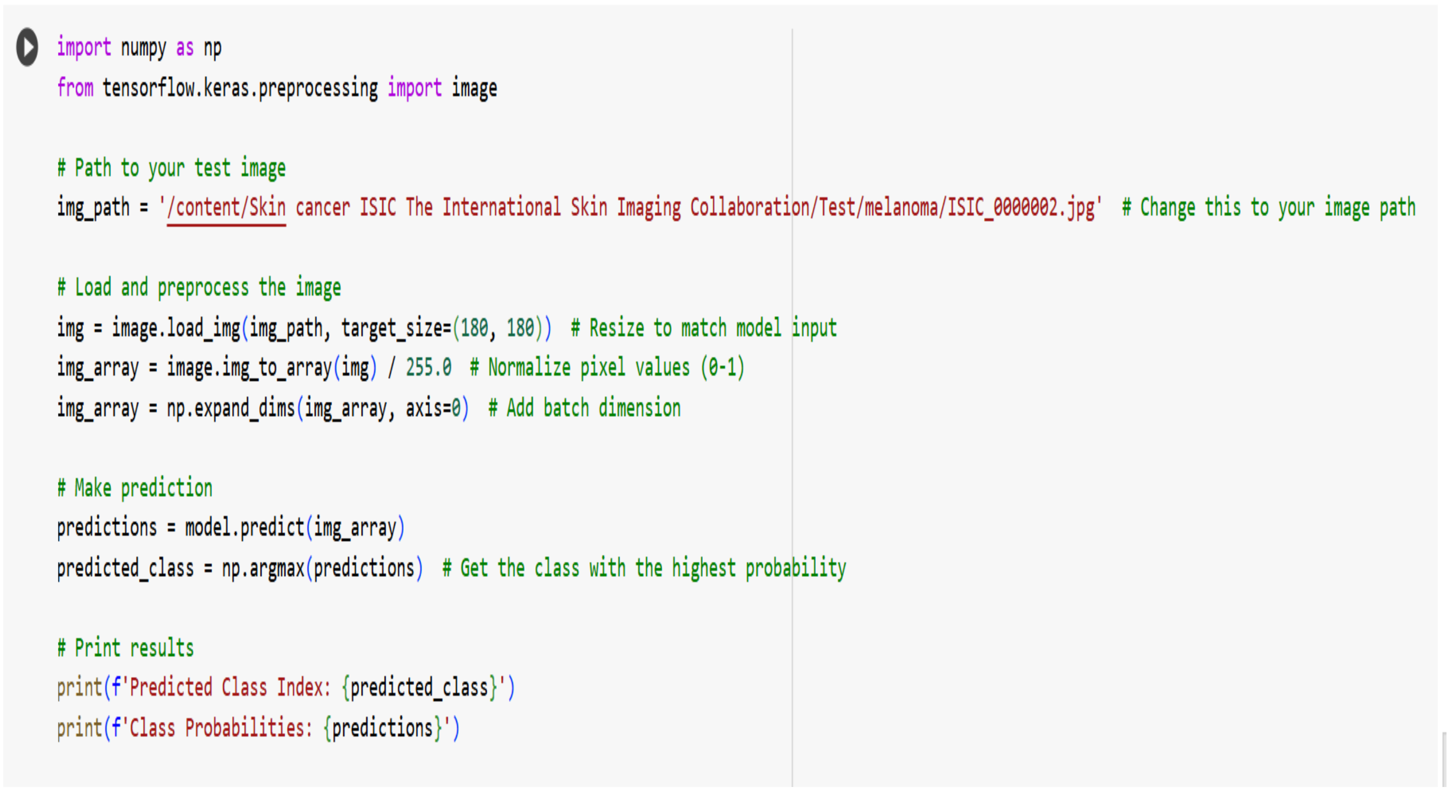


Figure 19 : Accuracy Rates

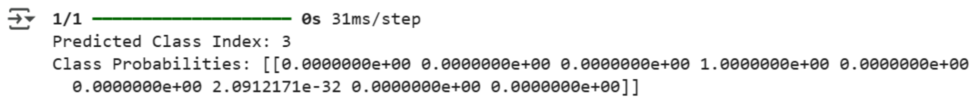
Final Results

Code to Test the Model on a Single Image



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

# Path to your test image
img_path = '/content/Skin cancer ISIC The International Skin Imaging Collaboration/Test/melanoma/ISIC_0000002.jpg'  # Change this to your image path

# Load and preprocess the image
img = image.load_img(img_path, target_size=(180, 180))  # Resize to match model input
img_array = image.img_to_array(img) / 255.0  # Normalize pixel values (0-1)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

# Make prediction
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions)  # Get the class with the highest probability

# Print results
print(f'Predicted Class Index: {predicted_class}')
print(f'Class Probabilities: {predictions}')
```

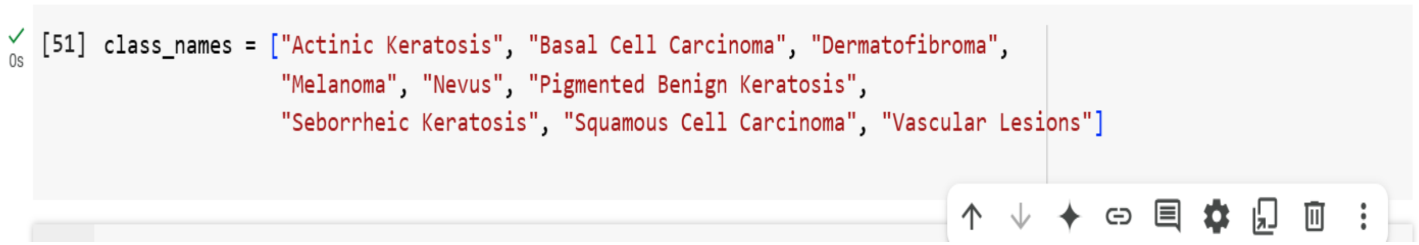
Understanding the Output



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
Predicted Class Index: 3
Class Probabilities: [[0.0000000e+00 0.0000000e+00 0.0000000e+00 1.0000000e+00 0.0000000e+00
  0.0000000e+00 2.0912171e-32 0.0000000e+00 0.0000000e+00]]
```

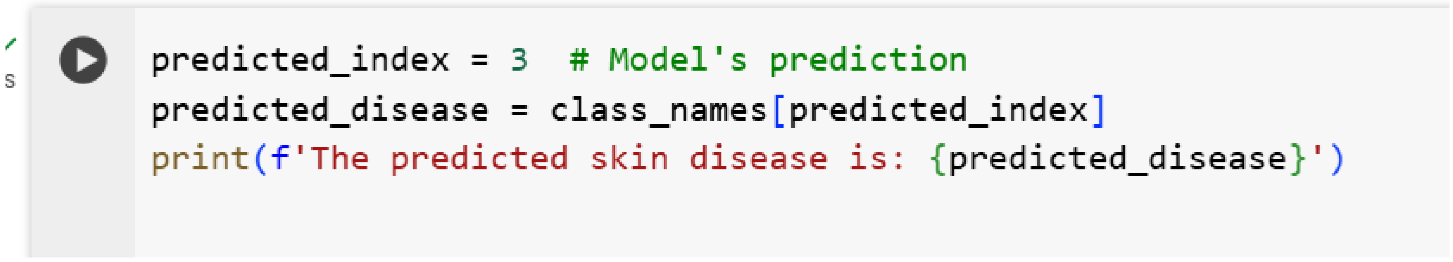
- This array represents the probabilities for each of the 9 classes (since our model classifies 9 different types of skin diseases).
- The values in the array correspond to the model's confidence level for each class.
- The highest probability is 1.0000000e+00 (or 100%) at index 3, meaning the model is fully confident that the image belongs to class 3.
- All other class probabilities are 0.0000000e+00, meaning the model does not consider them likely.

Check class labels



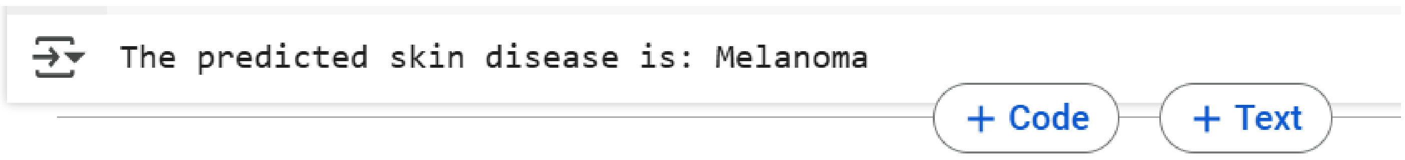
```
[51] class_names = ["Actinic Keratosis", "Basal Cell Carcinoma", "Dermatofibroma",
                   "Melanoma", "Nevus", "Pigmented Benign Keratosis",
                   "Seborrheic Keratosis", "Squamous Cell Carcinoma", "Vascular Lesions"]
```

Get the Predicted Disease Name



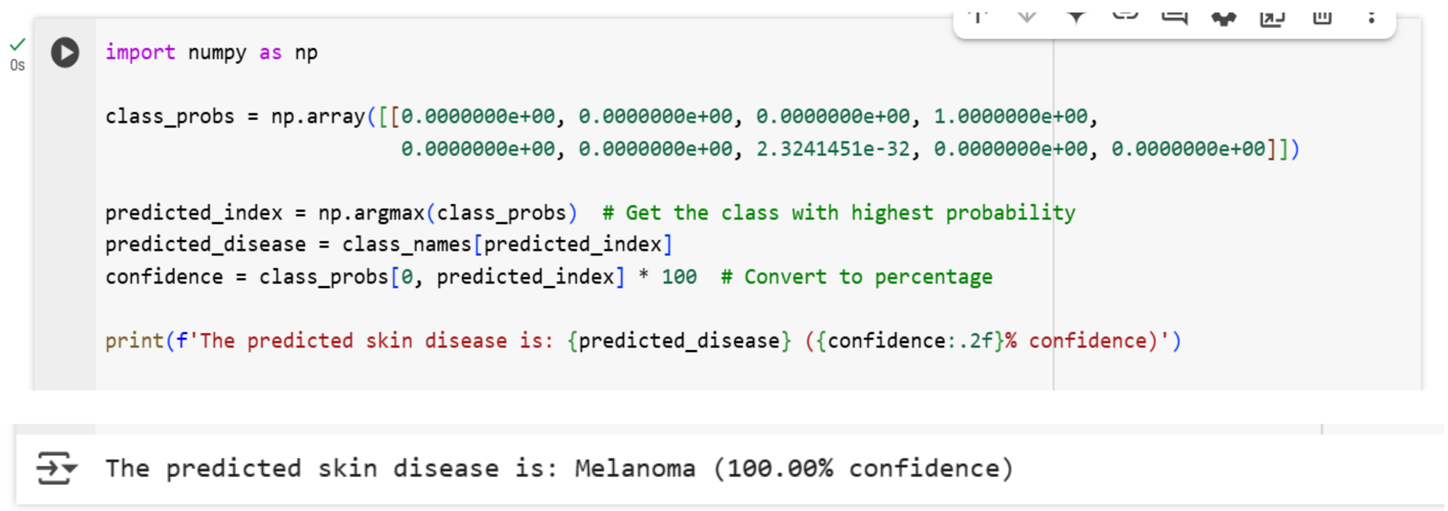
```
predicted_index = 3  # Model's prediction
predicted_disease = class_names[predicted_index]
print(f'The predicted skin disease is: {predicted_disease}')
```

Expected Output



```
The predicted skin disease is: Melanoma
```

Double-Check Model Confidence



```
import numpy as np

class_probs = np.array([[0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 1.0000000e+00,
                         0.0000000e+00, 0.0000000e+00, 2.3241451e-32, 0.0000000e+00, 0.0000000e+00]])

predicted_index = np.argmax(class_probs)  # Get the class with highest probability
predicted_disease = class_names[predicted_index]
confidence = class_probs[0, predicted_index] * 100  # Convert to percentage

print(f'The predicted skin disease is: {predicted_disease} ({confidence:.2f}% confidence)')
```

```
The predicted skin disease is: Melanoma (100.00% confidence)
```

Figure 20 : Final results

CHAPTER – 6

TESTING

SYSTEM TEST

System testing is a comprehensive process aimed at validating the overall functionality and performance of the integrated melanoma detection system. This type of testing evaluates whether all components—such as image preprocessing, model predictions, and output handling—work together seamlessly to deliver the expected results. The goal is to ensure that the software system meets its functional, non-functional, and security requirements, without any unexpected failures.

TYPES OF TESTS

Unit testing

Unit testing focuses on verifying that each individual module of the melanoma detection system, such as image loading and preprocessing algorithms, functions as expected. These tests validate that each component processes data correctly and adheres to the specifications defined during the design phase. It helps isolate any issues that might arise in a specific part of the system.

Integration testing

Integration testing ensures that various components of the melanoma detection pipeline work together without any conflicts. After verifying the individual functionality through unit testing, the system undergoes integration testing to confirm that modules such as the image input interface, model prediction engine, and output display communicate and work effectively as a complete system.

Functional test

Functional testing was executed to verify that the melanoma detection system performs as expected according to business and user requirements. This testing involved assessing whether the system successfully handles image inputs, processes them through the CNN model, and provides accurate outputs such as the classification of skin conditions (e.g., melanoma, benign lesions).

Functional testing is centered on the following items:

Valid Input : The system should accept valid input images. Invalid

Input : The system must reject invalid input, such as

unsupported image formats.

Functions : All identified functions must be tested.

Output : adhering to the expected outputs for each class.

Systems/Procedures: interfacing systems or procedures must be invoked.

System Test

System testing verifies that the entire system, when integrated, meets all the specified requirements. It is a high-level evaluation to ensure that the software delivers predictable and reliable results. For example, the accuracy of melanoma classification from the images, processing time, and response time of the system were all scrutinized to meet user expectations.

White Box Testing

White box testing was implemented to examine the internal logic of the code. This form of testing focuses on the paths, decisions, and code structure to ensure that the logic functions correctly. The purpose was to validate that the internal system behavior of the CNN and preprocessing pipeline is working without any flaws.

Black Box Testing

Black box testing focused purely on the output produced by the melanoma detection system. Without knowledge of the system's internal code, this test involved providing images and checking if the system generated correct predictions. The emphasis was on evaluating the accuracy and effectiveness of the model from an end-user perspective.

Unit Testing

Unit testing focuses on verifying that each individual module of the melanoma detection system, such as image loading and preprocessing algorithms, functions as expected. These tests validate that each component processes data correctly and adheres to the specifications defined during the design phase. It helps isolate any issues that might arise in a specific part of the system.

Test strategy and approach

Manual Testing: The system underwent manual field tests to simulate real-world conditions.

These tests helped ensure that the system could handle various images and classify them accurately under different scenarios.

Test objectives

The primary objectives were to confirm that the model could accurately detect melanoma and classify other skin conditions in images, while maintaining reliable performance even when faced with various image types (e.g., different lighting conditions, image quality)**.**

Features to be tested

- Image upload functionality
- Accuracy of melanoma detection
- Correct classification of skin conditions
- Overall system performance under different loads and image qualities.

Integration Testing

Integration testing ensures that various components of the melanoma detection pipeline work together without any conflicts. After verifying the individual functionality through unit testing, the system undergoes integration testing to confirm that modules such as the image input interface, model prediction engine, and output display communicate and work effectively as a complete system.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

User Acceptance Testing (UAT)

User Acceptance Testing (UAT) is the final phase of testing where the system is evaluated from the perspective of the end-users to ensure that it meets the specified business needs and user requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

SAMPLE TEST CASES

S.no	Description	Expected Outcome	Result
1.	Upload image	Image should be uploaded successfully and correctly.	Pass
2.	Detect Melanoma	Correctly identify melanoma in skin lesion images	Pass
3.	Classification of Skin Diseases	Accurate classification of various skin conditions.	Pass

Table 1 : Sample Test Cases

CHAPTER - 7

CONCLUSION

Throughout the project, several key methodologies were employed, including data preprocessing, augmentation, and addressing class imbalances, to ensure the model’s robustness and generalization. The trained model demonstrated strong performance in detecting melanoma, thereby offering a potential solution for assisting dermatologists in early detection and diagnosis, which is crucial for improving patient survival rates.

While the model achieved promising results, future improvements could focus on further optimizing the architecture to enhance accuracy and reduce overfitting. Additionally, extending the model's capabilities to accommodate a wider range of skin conditions could provide even greater utility in clinical settings.

In conclusion, the CNN-based melanoma detection model developed in this project demonstrates significant potential for improving diagnostic workflows in dermatology. By automating and augmenting the diagnostic process, this approach can help ease the strain on healthcare professionals and improve the early identification of skin cancer, contributing to better clinical outcomes and advancing the use of AI in medical imaging.

REFERENCES

[1]. Hossam Magdy Balaha et.aI “Skin cancer diagnosis based on deep transfer learning and sparrow search algorithm” ://doi.org/10.1007/s00521-022-07762-9(0123456789,-().,volV).

[2]. TWu Y, Chen B et.al Skin Cancer Classification with Deep Learning: A Systematic Review.Front. Oncol. 12:893972. doi:10.3389/fonc.2022.893972

[3]. Doaa Khalid Abdulridha Al-Saedi et.al ’Classification of Skin Cancer with Deep Transfer. Learning Method’ https://doi.org/10.53070/bbd.1172782

[4]. Viswanatha Reddy Allugunti “A machine learning model for skin disease classification using convolution neural network’. EISSN: 2707-6644

[5]. Gouda,W et.al . Detection of Skin Cancer Based on Skin Lesion Images Using Deep Learning. Healthcare 2022, 10, 1183.

[6]. M. Krishna Monika et.al ‘Skin cancer detection and classification using machine learning’ https://doi.org/10.1016/j.matpr.2020.07.3662214-7853/ 2020

[7]. Vijayalakshmi M M "Melanoma Skin Cancer Detection using Image Processing and Machine Learning" ISSN:

24566470, June2019, pp.780-784, URL:https://www.ijtsrd.com/papers/ijtsrd23936.pdf

[8]. Li-sheng Wei et.al Skin Disease Recognition Method Based on Image Color and Texture Features Hindawi Computational and

Mathematicalhttps://doi.org/10.1155/2018/8145713.

[9]. Basly H, Ouarda W, Sayadi FE, Ouni B, Alimi AM (2020) CNN-SVM learning approach based human activity recognition. In: Proceedings of international conference on image and signal processing. Springer, Cham, pp 271–281.

[10]. Brinker TJ, Hekler A, Enk AH, Berking C, Haferkamp S, Hauschild A, Weichenthal M, Klode J, Schadendorf D, Holland-Letz T, Kalle CV, Fröhling S, Schilling B, Utikal JS (2019) Deep neural networks are superior to dermatologists in melanoma image classification. Eur J Cancer 119:11–17

[11]. American Cancer Society - Key Statistics for Melanoma Skin Cancer (2021) https://www.cancer.org/cancer/types/melanoma-skin-cancer/about/key-statistics.html Accessed on 12th January 2021.

[12]. Chaturvedi SS, Tembhurne JV, Diwan T (2020) A multi-class skin Cancer classification using deep convolutional neural networks. Multimed Tools Appl 79(39):28477–28498.

[13]. Codella NC, Gutman D, Celebi ME, Helba B, Marchetti MA, Dusza SW, Halpern A (2018) Skin lesion analysis toward melanoma detection: a challenge at the 2017 international symposium on biomedical imaging (ISBI), hosted by the international skin imaging collaboration (ISIC). In IEEE 15th international symposium on biomedical imaging (ISBI 2018), 168-172

[14]. He K, Zhang X, Ren S, Su, J (2016) Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, 770–778. https://ieeexplore.ieee.org/document/7780459