International Centre for Education and Research (ICER)
VIT-Bangalore

Melanoma Detection Using Convolutional Neural Networks on Skin Lesion Images

CS7510 – PROJECT 2

REPORT

Submitted by

Jayanth Kumar – 24MSP3073

In partial fulfilment for the award of the degree of

POST GRADUATE PROGRAMME

INTERNATIONAL CENTRE FOR HIGHER EDUCATION AND RESEARCH

VIT BANGALORE

MARCH, 2024

International Centre for Education and Research (ICER)
VIT-Bangalore

BONAFIDE CERTIFICATE

Certified that this project report **"Melanoma Detection Using Convolutional Neural Networks on Skin Lesion Images"** is the bonafide record of work done by **"Jayanth Kumar – 24MSP3073"** who carried out the project work under my supervision.

Signature of the Supervisor	**Signature of Director**
Prof. S Ramya Mohana Krishnan	**Prof. Prema M**
Professor,	**Director,**
ICER	ICER
VIT Bangalore	VIT Bangalore.

Evaluation Date:

International Centre for Education and Research (ICER)
VIT-Bangalore

ACKNOWLEDGEMENT

I would like take this opportunity to express my sincere gratitude to our director of ICER **Prof. Prema M** for providing me an opportunity to widen my knowledge in field of computer science. It would take an extraordinary amount of time for me to personally recognise and appreciate each and every individual who has contributed to this project, whether directly or indirectly. A great deal of gratitude goes out to the numerous people whose cooperation and encouragement have made my job significantly simpler. **Prof. Ramya Mohanakrishnan**, ICER has provided me with an excellent opportunity to learn about my area of interest, without her continuous guidance and persistent help, this project would not have been a success for me. I am grateful to show my genuine appreciation to her. Taking this opportunity to offer my appreciation to our parents and friends for their support and encouragement throughout the project has been an honour.

Table of contents

Table of Contents

ABSTRACT

Skin cancer is one of the most common types of cancer worldwide, with early detection playing a critical role in improving survival rates. Traditional diagnostic methods, such as dermoscopy and biopsy, are time-consuming and require expert dermatologists. In recent years, deep learning has emerged as a powerful tool for automating skin lesion classification, enabling fast and accurate diagnosis. In this project, we leverage pre-trained model and custom CNN models, a lightweight convolutional neural network (CNN), to classify skin lesions into seven categories: melanoma (mel), nevus (nv), basal cell carcinoma (bcc), actinic keratoses (akiec), benign keratosis-like lesions (bkl), vascular lesions (vasc), and dermatofibroma (df). The model is trained on the HAM10000 dataset, a well-known benchmark dataset for skin lesion classification. Additionally, we implement a Gradio-based web application that allows users to upload an image and receive real-time predictions in the form of a probability distribution. The proposed system aims to assist dermatologists in early detection and classification of skin lesions, ultimately reducing the diagnostic burden and improving patient outcomes.

Keywords: skin leasion, convolutional neural network, Melonoma, Gradio (web-app)

List of Figures

List of tables

CHAPTER-1

INTRODUCTION

1.1 Project Background

Skin cancer is one of the most prevalent forms of cancer worldwide, and early detection plays a critical role in improving patient outcomes. Dermatologists rely on visual examination and dermoscopic images to diagnose various types of skin lesions, but manual diagnosis can be subjective and time-consuming.

To address this challenge, this project aims to develop a machine learning-based skin lesion classification system using deep learning techniques. By leveraging a pre-trained convolutional neural network (CNN), the model is trained to classify skin lesions into different categories based on medical image data. The trained model is then deployed as a Gradio web application, allowing users to upload an image and receive classification predictions with probability scores displayed in a bar chart.

This system is designed to assist medical professionals and individuals by providing an accessible, AI-powered tool for preliminary skin lesion analysis. While it is not a replacement for professional medical diagnosis, it serves as a supportive tool to enhance early detection efforts and improve healthcare accessibility.

1.2 Scope of Study

This study focuses on developing a deep learning-based skin lesion classification model using a pre-trained convolutional neural network (CNN). The model is trained on medical image datasets to classify skin lesions into different categories. The project includes data preprocessing, model training, and evaluation to ensure reliable performance.

The trained model is deployed as a Gradio web application, allowing users to upload skin lesion images and receive classification predictions with probability scores. The study does not cover real-time clinical diagnosis or treatment recommendations but serves as a supportive tool for preliminary analysis.

1.3 Problem Statement

Skin cancer, particularly melanoma, is one of the deadliest forms of cancer, and early detection is crucial for effective treatment. However, traditional diagnostic methods rely on expert dermatologists, which can be time-consuming and prone to human error. This project aims to develop an AI-powered deep learning model for automated melanoma classification using skin lesion images. By leveraging Convolutional Neural Networks (CNNs) and advanced image processing techniques, the system will enhance diagnostic accuracy, reduce the burden on healthcare professionals, and enable early detection, ultimately improving patient outcomes.

1.4 Need of Study

- **Rising Cases of Skin Cancer** – Skin cancer is one of the most common cancers globally, and early detection significantly improves treatment outcomes.
- **Challenges in Manual Diagnosis** – Dermatologists rely on visual examination, which can be subjective and time-consuming.
- **Role of AI in Medical Imaging** – Deep learning models can assist in automating and enhancing accuracy in skin lesion classification.
- **Improving Accessibility** – A web-based AI tool allows for quick preliminary screening, especially in areas with limited medical resources.
- **Support for Medical Professionals** – The system can act as a second opinion, helping dermatologists make more informed decisions.
- **Non-Invasive Early Detection** – Provides a non-invasive method to assess potential skin lesions before seeking medical consultation.

1.5 Objectives

The objective of this project is to develop a deep learning-based melanoma classification system that can accurately detect and classify skin lesions from dermoscopic images using advanced Convolutional Neural Networks (CNNs). This system aims to assist dermatologists in early melanoma detection, improving diagnostic efficiency and reducing the risk of misdiagnosis.

To achieve this, the project will:

- **Leverage Transfer Learning**: Utilize pre-trained models such as **EfficientNet-B7, ResNet-50** and **custom CNN** to enhance feature extraction, reducing training time and improving model performance.

- **Address Class Imbalance**: Since melanoma is rarer than benign lesions, implement class balancing strategies such as weighted loss functions, synthetic data generation (SMOTE), or imbalanced-learn sampling to prevent biased predictions.

- **Evaluate Performance Using Robust Metrics**: Assess the model using accuracy, precision, recall, F1-score, AUC-ROC, and confusion matrices to measure effectiveness across different lesion types.

- **Compare Model Architectures**: Experiment with multiple CNN architectures (**EfficientNet, ResNet, MobileNet, etc.**) to identify the best-performing model for melanoma classification.

- **Deploy the Model for Real-World Use (Future Scope)**: Explore possibilities for integrating the trained model into **a web or mobile-based application** for dermatologists and healthcare professionals.

This project ultimately aims to develop a highly accurate, efficient, and generalizable deep learning model that can aid in the early detection of melanoma, potentially improving patient outcomes through timely diagnosis and intervention.

CHAPTER-2

LITERATURE REVIEW

2.1 Previous Work

Researchers had undertaken an exhaustive exploration of various algorithms employed for feature selection and melanoma prediction, as elaborated in [1]. These predictive frameworks were intricately devised for diagnosing melanoma, incorporating real-world data-gathering approaches. The development of the model hinged on a series of guidelines formulated in collaboration with domain experts. Noteworthy is the substantial investment necessitated by the need for swift image processing and the acquisition of nuanced insights to fine-tune the optimal rule set. Interestingly, many researchers chose to incorporate deep learning methodologies into their system. In this study, the authors presented a novel concept focused on employing a one-class deep neural network learning framework. Utilizing advanced deep learning techniques and publicly available dermoscopic images, the system significantly improved melanoma detection. Initially, the image classification utilized a deep convolutional neural network, integrating feature extraction methods. Subsequently, these extracted features were inputted into a pre-trained model.

The authors of [2] presented a novel deep learning-based method designed to enhance the accuracy of classification systems. The method employs a multi-scale decomposition model in the preprocessing phase, utilizing texture as input for the CNN to eliminate separate texture extraction and feature selection. The experimental findings demonstrate that the proposed approach attains superior accuracy when contrasted with other established algorithms and methodologies documented in the existing literature.

In another research study [3], CNN was employed to detect malignant and benign lesions utilizing the ISIC2018 dataset, comprising 3533 images on a variety of skin lesions, ranging from benign and malignant tumors to nonmelanocytic and melanocytic growths. Initially, the images underwent enhancement using ESRGAN. During preprocessing, augmentation, normalization, and resizing techniques were applied to the images. The CNN method was utilized to classify the skin lesion images, based on aggregated results from multiple iterations. Furthermore, various transfer learning models, including ResNet50, InceptionV3, and Inception ResNet, underwent fine-tuning. The custom CNN model achieved an accuracy of 83.2%, which was comparable to the pre-trained models: Resnet50 (83.7%), InceptionV3 (85.8%), and Inception Resnet (84%).

In the research cited in [4], a deep learning model was developed to detect skin cancer using the HAM10000 dermoscopic image database that includes 513 BCC, 790 benign, 327 actinic and intraepithelial carcinoma (AKIEC), and 115 dermatofibroma events. In this research, a CNN was created to detect benign and malignant groups. AlexNet was utilized as the pre-trained model. This model directly processes raw images, learning crucial features for classification without lesion segmentation or manual feature extraction. It achieved an accuracy of 84%, specificity of 88%, an area under the receiver operating characteristic (ROC) curve of 0.91, and sensitivity of 81%, with a confidence score threshold of 0.5. Deep learning was utilized to identify malignant and benign tumors using skin images of the RGB channel. A combination of lesion segmentation and classification was employed to detect malignant and benign tumors. For segmentation, U-Net was utilized, and an algorithm was used for classification.

2.2 Comparative Work

S.No	**Title of the Paper**	**Authors**	**Journal Name/Conference Title & Publisher**	**Year of Publication**	**Methodology used**	**Performance of the proposed algorithm**
1.	Skin lesion classification using ensembles of multi-resolution EfficientNets with meta data	Nils Gessert, Maximilian Nielsen, Mohsin Shaikh, RenéWerner, Alexander Schlaefer	Elsevier	2020	EfficientNets, SENet, and ResNeXt WSL	95.16%
2.	Skin lesion classification of dermoscopic images using machine learning and convolutional neural network	Bhuvaneshwari Shetty, Roshan Fernandes, Anisha P. Rodrigues, Rajeswari Chengoden, Sweta Bhattacharya & Kuruva Lakshmanna	Scientific Reports	2022	Custom CNN and Pre-trained models	91.8%
3.	Skin Lesion Classification Using a Deep Ensemble Model	Su Myat Thwin and Hyun-Seok Park	MDPI	2024	VGG16, Inception-V3, and ResNet-50	93.2%
4.	Skin lesion classification using machine learning approach: A survey	Adnan Afroz; Razia Zia; Andres Ortiz Garcia; Muhammad Umar Khan; Umair Jilani; Khawaja Masood Ahmed	IEEE Xplore	2022	Ensemble techniques	92.48%

Table 2.2: Comparative work of previous study

CHAPTER-3

METHODOLOGY

3.1 Design/Architecture Diagram

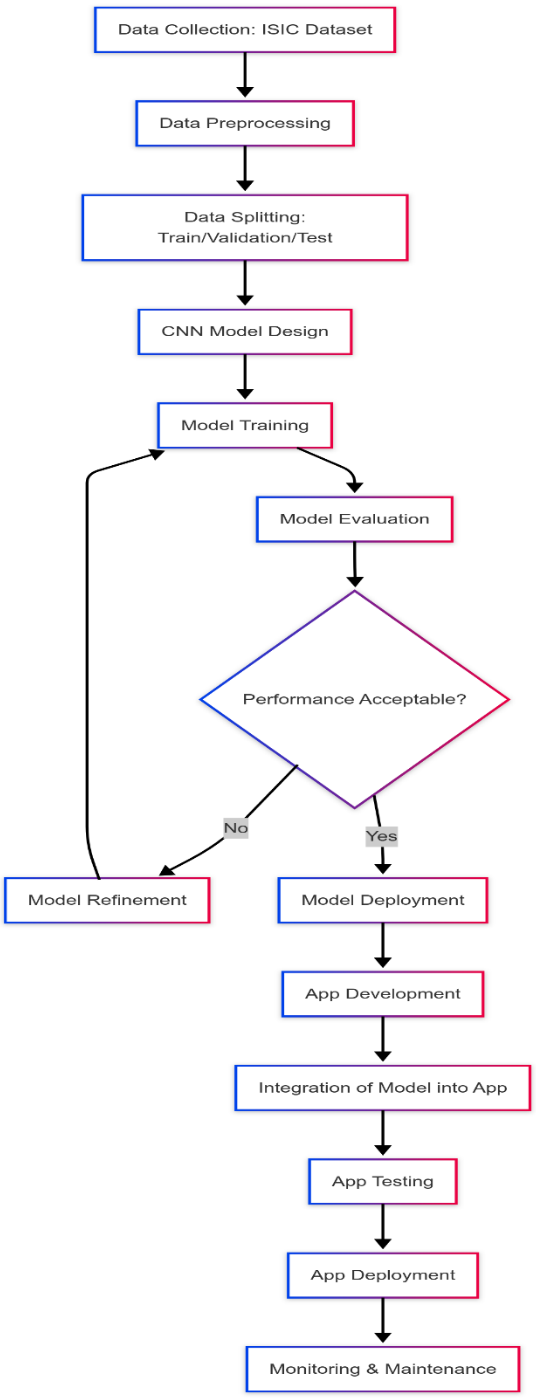


Figure 3.1: Flowchart

3.2 Dataset

The **HAM10000** ("Human Against Machine with 10,000 training images") dataset is a large collection of skin lesion images designed for machine learning research in dermatology, particularly for skin cancer classification. It was published to support automated skin lesion diagnosis using deep learning models.

Key Features of the Dataset

- **Total Images**: 10,015 high-resolution dermoscopic images.
- **Lesion Categories**: Covers 7 different types of skin lesions.
- **Data Source**: Collected from different populations and different datasets to ensure diversity and reduce bias.
- **Annotation**: Each image is labelled with a corresponding diagnosis, verified by clinical experts.

Label	Lesion Type	Description
nv	Melanocytic nevi	Benign moles, common skin lesions
mel	Melonoma	A dangerous type of skin cancer that can spread
bkl	Benign keratosis	Non-cancerous growths, including solar lentigines and seborrheic keratoses
bcc	Basil cell carcinoma	A common, slow-growing type of skin cancer
akiec	Actinic keratoses	Precancerous lesions that can develop into squamous cell carcinoma
vasc	Vascular lesions	Blood vessel-related skin abnormalities, such as angiomas
df	Dermatofibroma	A benign skin lesion caused by an overgrowth of fibrous tissue

Table 3.2: Different lesion labels

3.3 Dataset Structure

The dataset is provided in two image formats:

- **JPEG images**: Stored in a folder structure.
- **CSV Metadata file**: Includes image filenames, corresponding lesion types, patient IDs, and other attributes.

Key metadata fields in the CSV file:

- image_id – Name of the image file.
- dx – Diagnosis (lesion type, e.g., ‘mel’, ‘nv’).
- dx_type – Method used to confirm the diagnosis (e.g., histopathology, dermatoscopy).
- age – Age of the patient.
- sex – Gender of the patient.
- localization – Body area where the lesion is located.

3.4 Image preprocessing

The proposed work applied the following image pre-processing steps.

3.4.1 Image Resizing

1. All the images in the folder are resized to 220*220 before processing into different machine learning models.

2. For the customized CNN model, images are scaled to 96 × 96 with a depth of 3 to speed up the process. Then we have converted the images into a NumPy array to get the value of each pixel of the image. Then we normalized the pixel values to a range of 0–1. The LabelBinarizer class allows us to input class labels that are in string form in the dataset, convert those class labels into one-hot encoded vectors, and then convert them back into a human-readable form from the integer class label prediction of Keras CNN.

3.4.2 Data Augmentation

1. Data augmentation is a technique for generating new “data”. To train the machine learning models, the proposed method used Horizontal Flip augmentation i.e., shifting all pixels of an image in a horizontal direction. As a result, models with data augmentation are more likely to learn more differentiating characteristic features than models without data augmentation. We have 200 images from each class after augmentation and trained the model with a dataset of 200*7 images. Figure 3 depicts the sample images after Horizontal Flip augmentation.

2. In the CNN model since we are working with a finite number of data points (each class has 200 images), we have applied random transformations (rotations, shearing, etc.) to

train. Each epoch had the same number of images as the original images. Overfitting is also avoided by data augmentation.

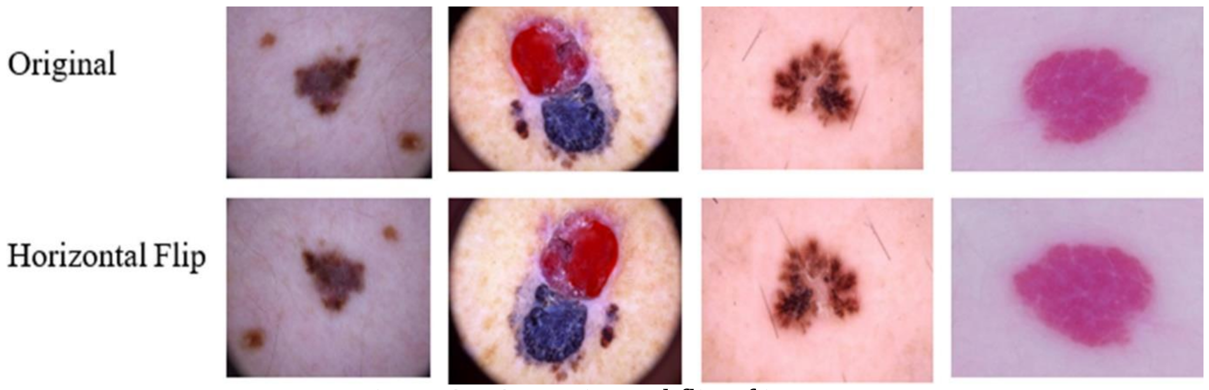


Figure 3.4.2: Horizontal flip of images

3.4.3 Feature Extraction

Global Feature Descriptors are used to quantify an image in its entirety. These don't have the concept of interest points and thus, take the entire image for processing. The color of the skin lesion image is quantified using a Colour Histogram. The shape of the skin lesion is quantified using Hu Moments. The texture of the skin lesion is quantified using a Haralick Texture. These features are chosen because the color, shape, and texture are the only properties that dominate in the lesion zone. The feature extraction experiment works with one image at a time, extracting three global features, concatenating them into a single global feature, and saving it in h5 format with its label.

3.4.4 Data Splitting

The OpenCV application was used to validate the machine learning models. The total number of images obtained from the HAM1000 dataset for Machine Learning model training is 700 (100 images from each class), 560 of which images represent 80% for training and 140 images represent 20% for testing. Due to the massive class imbalance revealed by the data set, this was necessary. After augmenting the dataset, 1400 images (200 images from each class) are used for ML model training, with 1120 images accounting for 80% of the training and 280 images accounting for 20% of the testing.

3.4.5 Convolutional Neural Network

In contrast to a standard neural net, a CNN learns detailed patterns by applying filters to the raw pixels of an image. To build a CNN, Tensorflow and Keras libraries were used to build and implement the model in Python 3.7.9. A high-level overview of CNN Architecture is shown in Fig 3.4.5. The layers and hyperparameters employed in the network are summarized in Table 3.4.5.

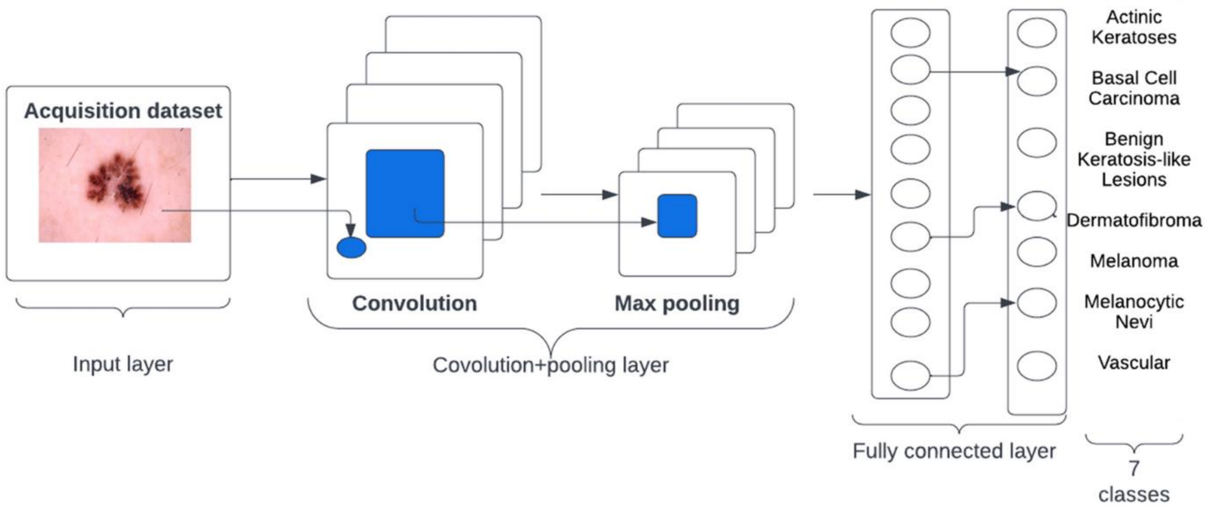


Figure 3.4.5: CNN Architecture

Layers	Hyperparameters
Conv2D	16 filters, 3 × 3 filter size, ReLU activation, same padding, followed by batch normalization
MaxPool2D	3 × 3 pool size to reduce image spatial dimensions quickly from 96 × 96 to 32 × 32
Conv2D	64 filters, 3 × 3 filter size, ReLU activation, same padding
Conv2D	128 filters, 3 × 3 filter size, ReLU activation, following the same padding, batch normalization is performed
MaxPool2D	2 × 2 pool size
Conv2D	128 filters, 3 × 3 filter size, ReLU activation, same padding followed by batch normalization
MaxPool2D	2 × 2 pool size
Flatten (Core Layer)	--
Dense	2048 Units, ReLU sctivation, and batch normalization
Dense	7 Units, softmax activation

Table 3.4.5: CNN layers and hyperparameters

3.4.6 Model hyperparameters

To get a better model evaluation, certain common hyperparameter values are used. The hyperparameter values utilized in the CNN model are highlighted in Table 3.4.6. The following section explains why the values of the hyperparameters were chosen in the proposed work: Optimizer: Adam is the most widely used optimization method for training deep neural networks today because it is simple to use, computationally efficient, and effective when dealing with enormous amounts of data and parameters. Loss Function: The Multi-Class calculates the loss value using the "categorical cross-entropy" loss function. Epochs: The epoch count is 50. Found that 50 epochs result in a model with low loss and no overfitting to the training set through experimentation (or not overfitted as best as we can). Batch Size: Several early tests with batch sizes of 5, 10, 20, and 40 found that batch size 32 produced the best results. Learning Rate: The rate of learning is initially set to 0.001. The "step" we take along the gradient is controlled by the learning rate. The smaller the value, the smaller the step, and the larger the value, the bigger the step.

Hyperparameter	**Value**
Optimizer	Adam
Loss function	Sparse_categorical_crossentropy
Epochs	50
Batch size	32
Learning rate	0.001-0.00001

Table 3.4.6: CNN model hyperparameters

CHAPTER-4

TOOLS AND PACKAGES

4.1 Oversampling using Imbalanced-learn package

Oversampling is a technique used to handle imbalanced datasets by increasing the number of samples in the minority class. The Imbalanced-Learn (imblearn) package provides efficient oversampling methods to balance the dataset before training a machine learning model.

4.2 VGG19

VGG-19 is a deep convolutional neural network with 19 weight layers, comprising 16 convolutional layers and 3 fully connected layers. The architecture follows a straightforward and repetitive pattern, making it easier to understand and implement.

The key components of the VGG-19 architecture are:

1. **Convolutional Layers**: 3x3 filters with a stride of 1 and padding of 1 to preserve spatial resolution.
2. **Activation Function**: ReLU (Rectified Linear Unit) applied after each convolutional layer to introduce non-linearity.
3. **Pooling Layers**: Max pooling with a 2x2 filter and a stride of 2 to reduce the spatial dimensions.
4. **Fully Connected Layers**: Three fully connected layers at the end of the network for classification.
5. **Softmax Layer**: Final layer for outputting class probabilities.

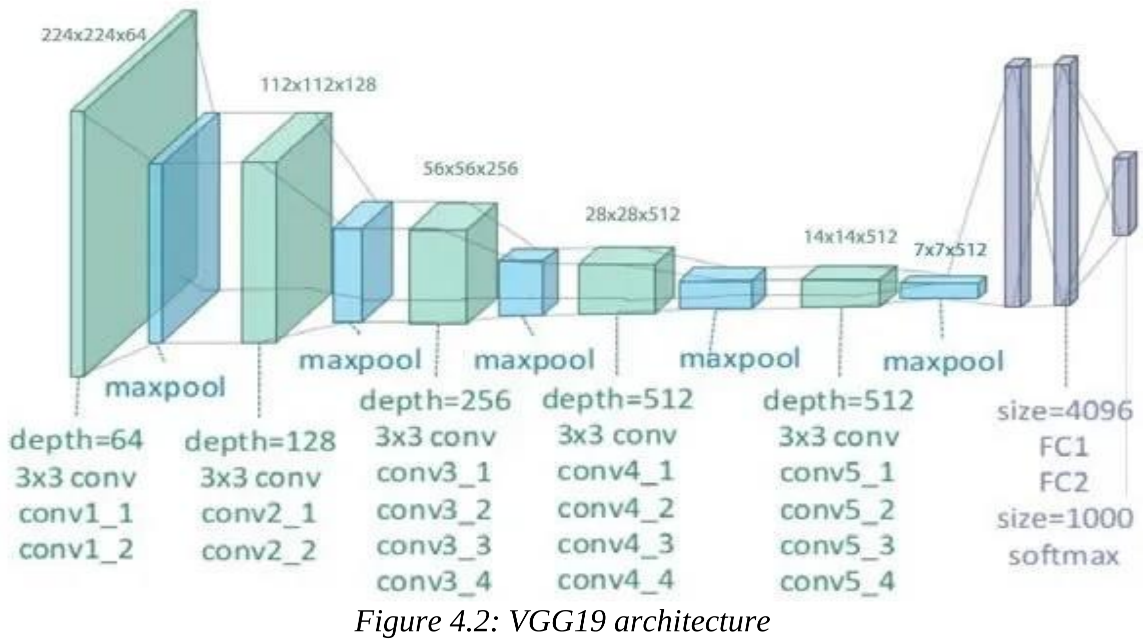


Figure 4.2: VGG19 architecture

4.3 ResNet50

ResNet-50 is a deep convolutional neural network (CNN) architecture distinguished by its use of residual learning. Comprising 50 layers, it integrates residual blocks that include shortcut connections. Each residual block typically consists of a few convolutional layers. By enabling direct paths for gradients to flow, ResNet-50 significantly enhances training efficiency and performance, making it highly effective for image classification tasks

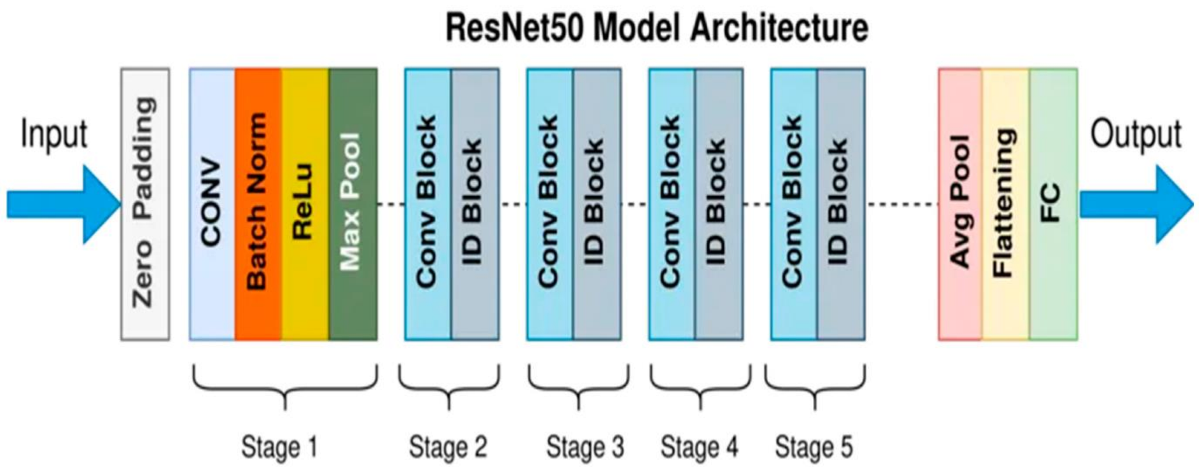


Figure 4.3: ResNet50 architecture

4.4 EfficientNetB7

Despite the fact that many of the models are computationally demanding, their application in training the ImageNet dataset has expanded in complexity and success. One of the most advanced models, the EfficientNet model, uses 66 M parameters to classify the ImageNet dataset with an accuracy rating of 84.4 percent. Therefore, it may be considered a set of CNN models. The eight models that make up the EfficientNet model range in size from B0 to B7; while accuracy increases greatly with the number of models, no discernible rise in the number of predicted parameters occurs. This novel activation function called the Leaky ReLu activation function, is used by the EfficientNet in place of the Rectifier Linear Unit (ReLu). When breadth, resolution, and depth are consistently scaled at smaller model sizes compared to other cutting-edge models, EfficientNet yields more efficient outputs.

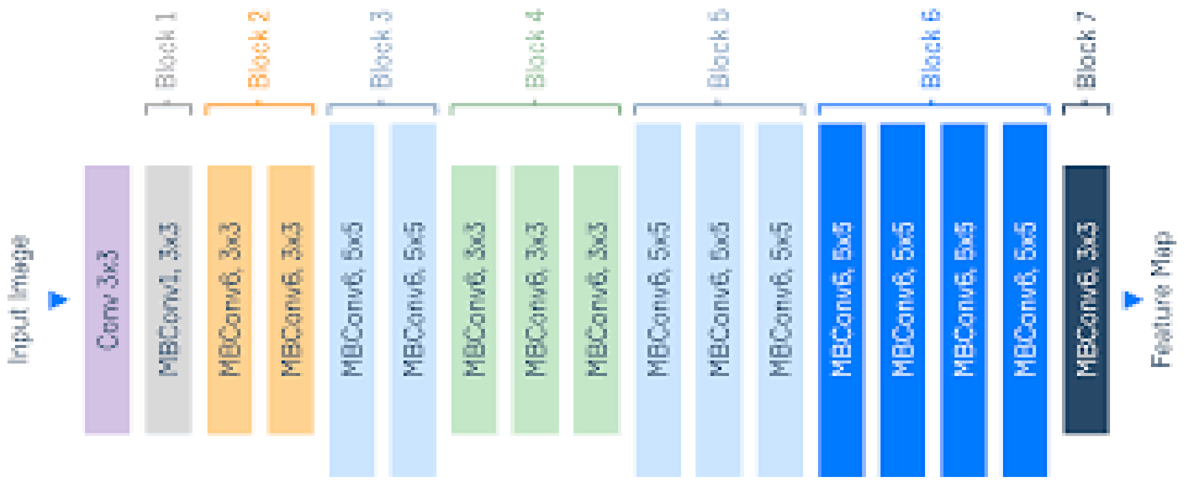


Figure 4.4: EfficientNetB7 architecture

4.5 MobileNet

A lightweight convolutional neural network (CNN) architecture, MobileNetV2, is specifically designed for mobile and embedded vision applications. Google researchers developed it as an enhancement over the original MobileNet model. Another remarkable aspect of this model is its ability to strike a good balance between model size and accuracy, rendering it ideal for resource-constrained devices.

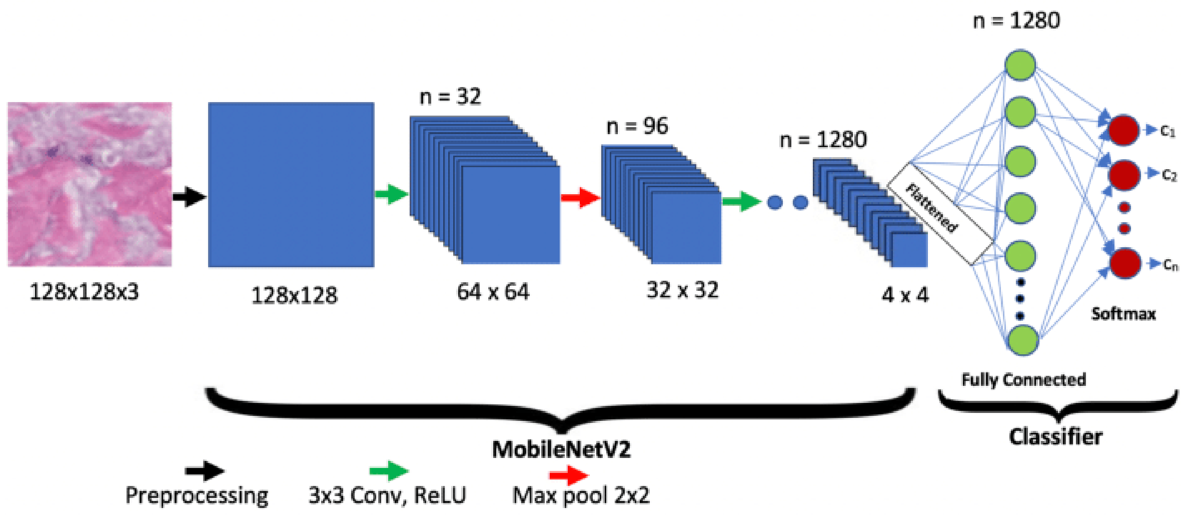


Figure 4.5: MobileNet architecture

4.6 Gradio

Gradio is an open-source python library which allows you to quickly create easy to use, customizable UI components for your ML model, any API, or any arbitrary function in just a few lines of code. You can integrate the GUI directly into your Python notebook, or you can share the link to anyone.

Chapter-5

RESULTS AND DISCUSSIONS

5.1 Data Preprocessing

Data preprocessing techniques include handling missing values in metadata csv and balancing the class labels using imbalanced-learn library. Age was the only variable which had missing value and it was handled manually using ‘fillna’. Apart from missing values there was huge imbalance among different class labels. Nv (Melanocytic nevi) class had huge count initially, in order to balance the other classes, we had to use RandomOverSampler technique to over-sample other classes to match the count of class nv.

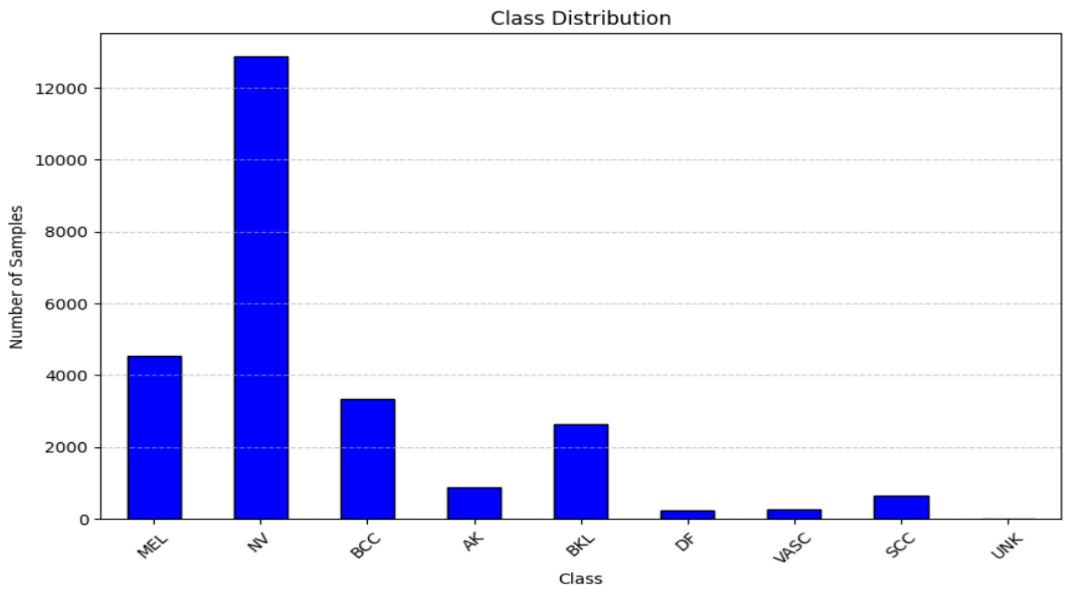


Figure 5.1a: Class distribution before balancing the data

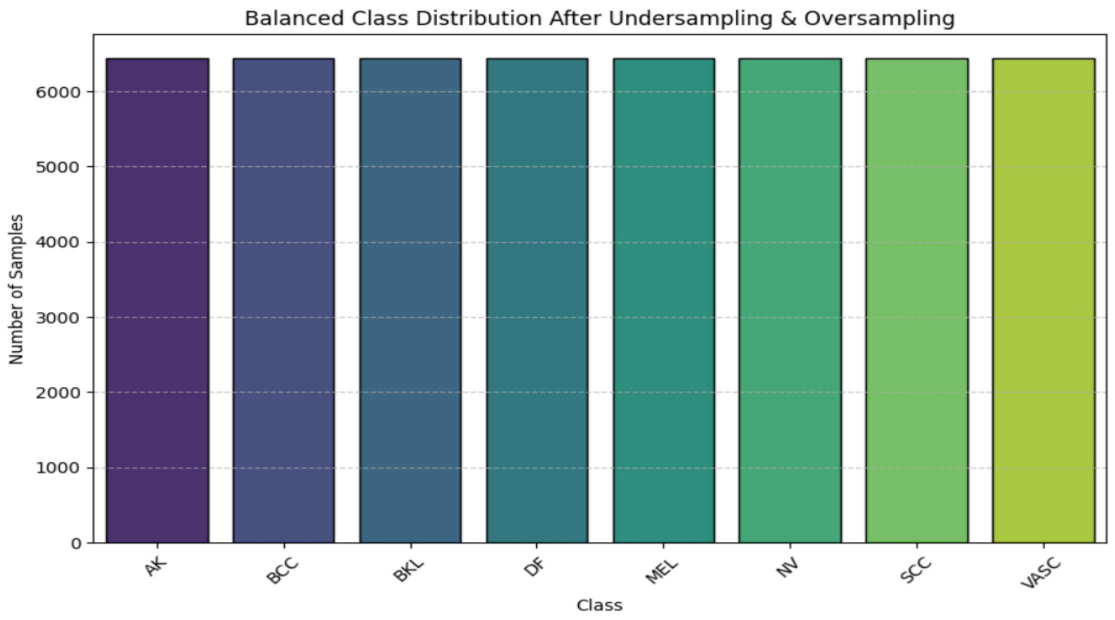


Figure 5.1b: Class distribution after balancing the data

5.2 Exploratory Data Analysis

Since the we are dealing with large sets of images, minimal EDA had been performed on the data.

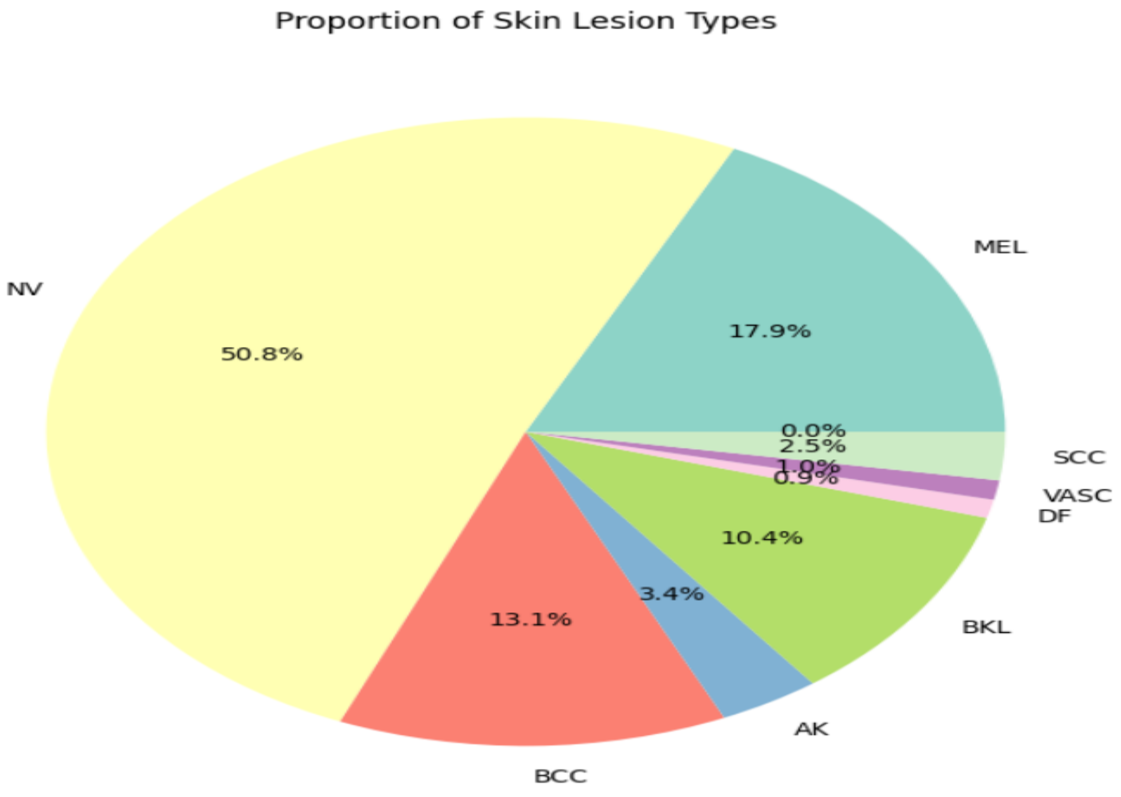
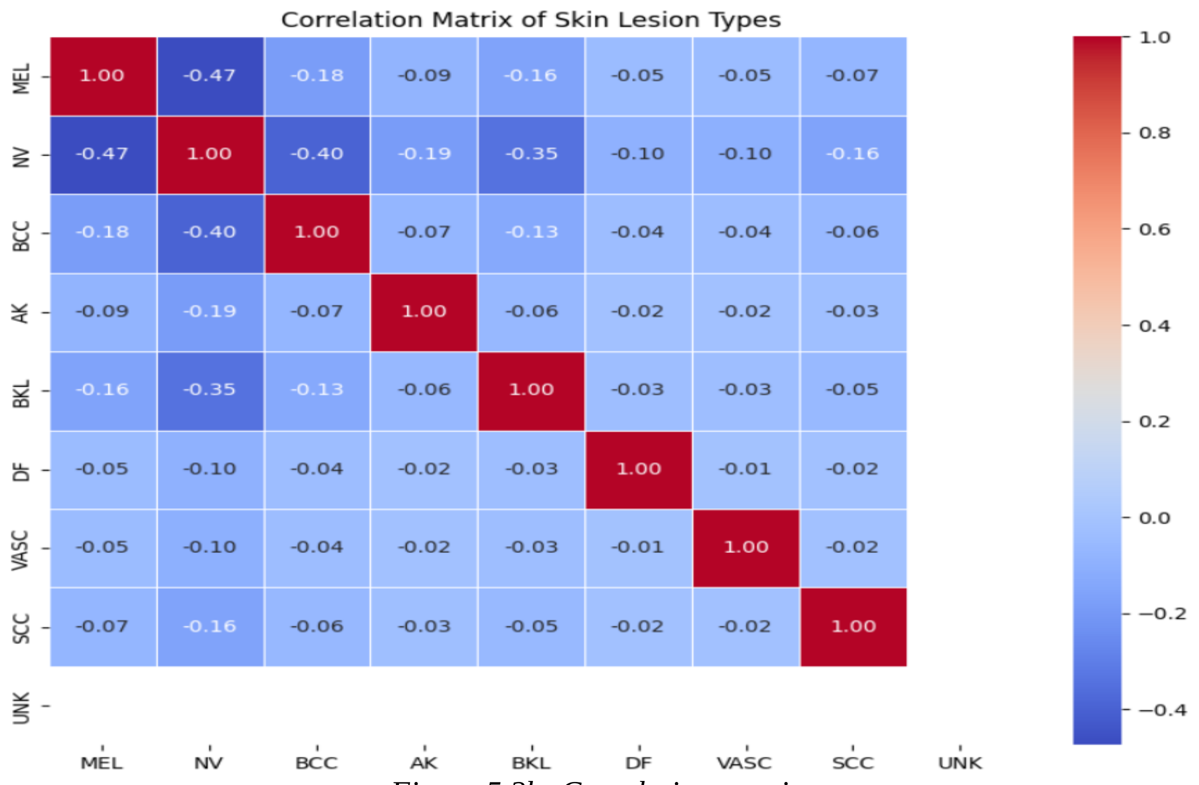


Figure 5.2a: Proportion of lesion types



	MEL	NV	BCC	AK	BKL	DF	VASC	SCC	UNK
MEL	1.00	-0.47	-0.18	-0.09	-0.16	-0.05	-0.05	-0.07	
NV	-0.47	1.00	-0.40	-0.19	-0.35	-0.10	-0.10	-0.16	
BCC	-0.18	-0.40	1.00	-0.07	-0.13	-0.04	-0.04	-0.06	
AK	-0.09	-0.19	-0.07	1.00	-0.06	-0.02	-0.02	-0.03	
BKL	-0.16	-0.35	-0.13	-0.06	1.00	-0.03	-0.03	-0.05	
DF	-0.05	-0.10	-0.04	-0.02	-0.03	1.00	-0.01	-0.02	
VASC	-0.05	-0.10	-0.04	-0.02	-0.03	-0.01	1.00	-0.02	
SCC	-0.07	-0.16	-0.06	-0.03	-0.05	-0.02	-0.02	1.00	
UNK									

Figure 5.2b: Correlation matrix

5.3 Model accuracy and loss graphs

- The suggested model's accuracy is denoted by the percentage of correct predictions it makes.
- The loss curve illustrates the model's error, providing insights into its performance. It quantifies how effectively or ineffectively the model is performing.
- The model underwent validation for 50 epochs. The blue curve depicts the performance on the training data, while the orange curve represents the performance on the validation data.

5.3.1 CNN loss and accuracy without Data Augmentation

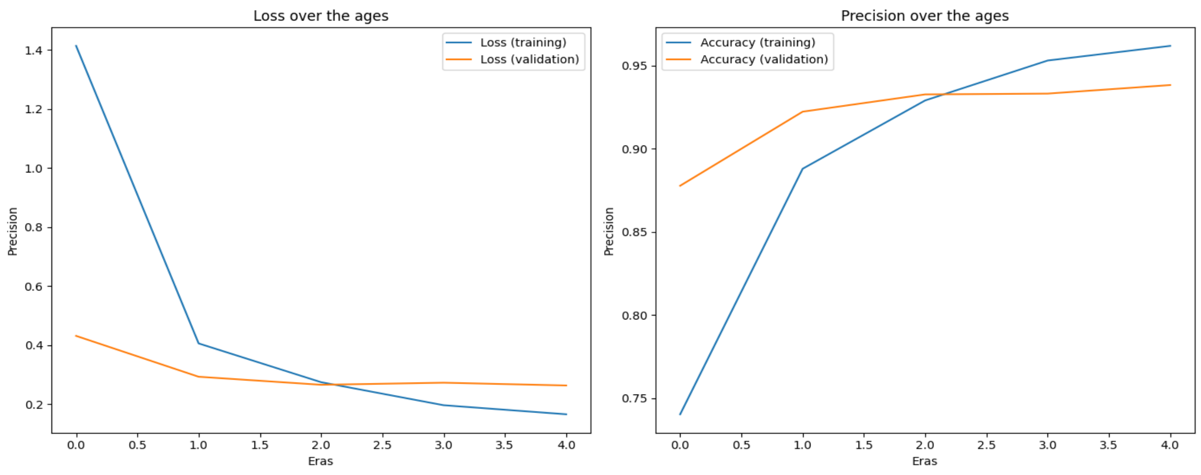


Figure 5.3.1: Loss and accuracy before augmenting

In Figure 5.3.1, the y-axis corresponds to the precision rate, while the x-axis shows how many epochs there are in total. The blue line on the graph signifies the training score, and the orange line represents the epoch count. At this juncture, the accuracy stands at a higher value (0.83), and further increasing the number of training instances may lead the CNN to underfit the dataset.

5.3.2 CNN loss and accuracy with Data Augmentation

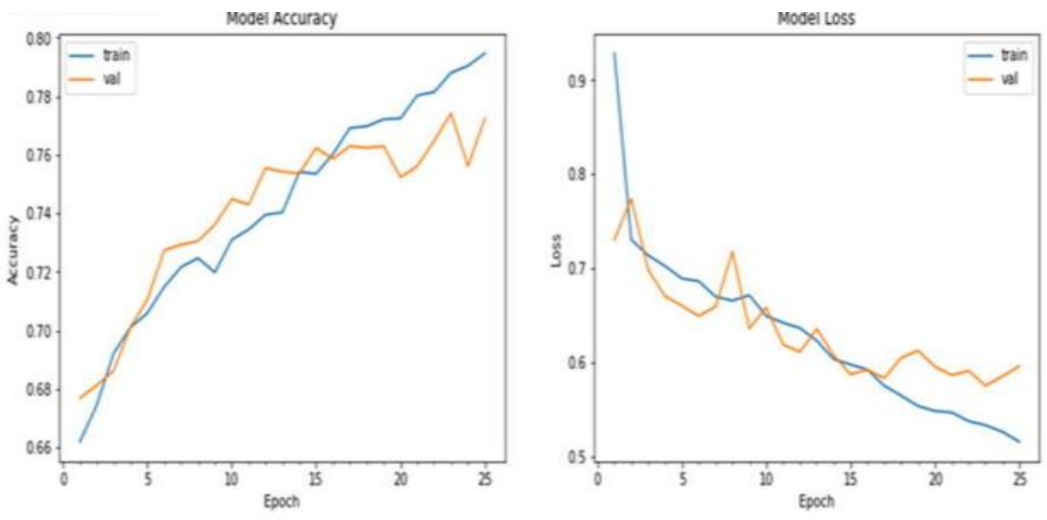


Figure 5.3.2: Loss and accuracy after augmenting

In Figure 5.3.2, the y-axis illustrates the precision rating, while the x-axis shows how many epochs there are in total. The blue line in the chart corresponds to the training score, and the orange line signifies the epoch count. Notably, the accuracy reaches a higher value at this juncture (0.99), and there is a possibility that further increasing the number of training instances may lead the CNN to underfit our dataset.

5.4 ROC Curves

- The Receiver Operating Characteristics (ROC) curve illustrates the true positive rate versus the false positive rate across various categorization thresholds.

- In the subsequent graphs, the orange curves depict the performance of our model under different decision-rule settings, effectively identifying genuine positive values from fake positive ones. The ROC accuracy, represented by the blue line, is calculated based on the distance between the model performance curve and the linear.

5.4.1 ROC curve without Data Augmentation

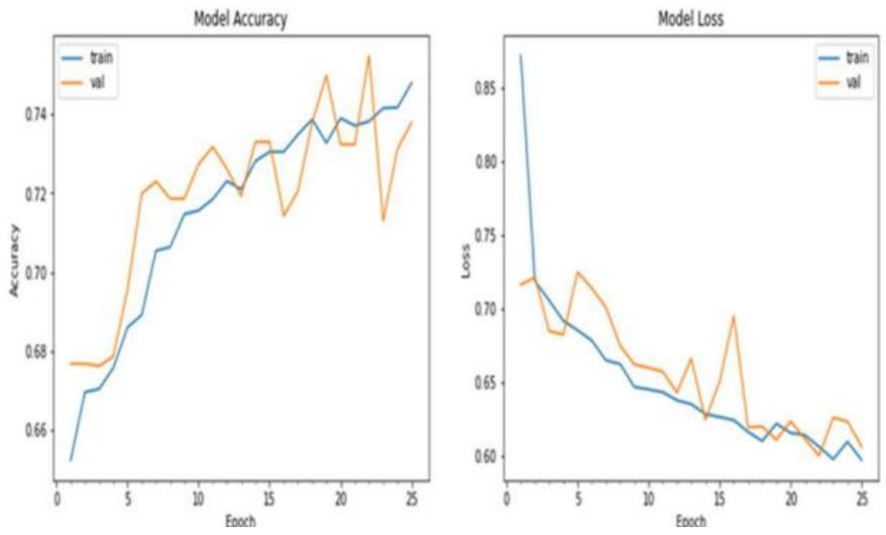


Figure 5.4.1: ROC before augmenting

Figure 5.4.1 above illustrates the true positive score at different decision rules on the y-axis and the false positive rate score on the x-axis, representing thresholds. The model's effectiveness is depicted by the orange line, which forms the Receiver Operating Characteristics (ROC) curve, while the baseline is represented in blue. The computation of the ROC score relies on the area beneath the ROC curve, and in this case, the model's ROC value is 0.8672.

5.4.2 ROC curve with Data Augmentation

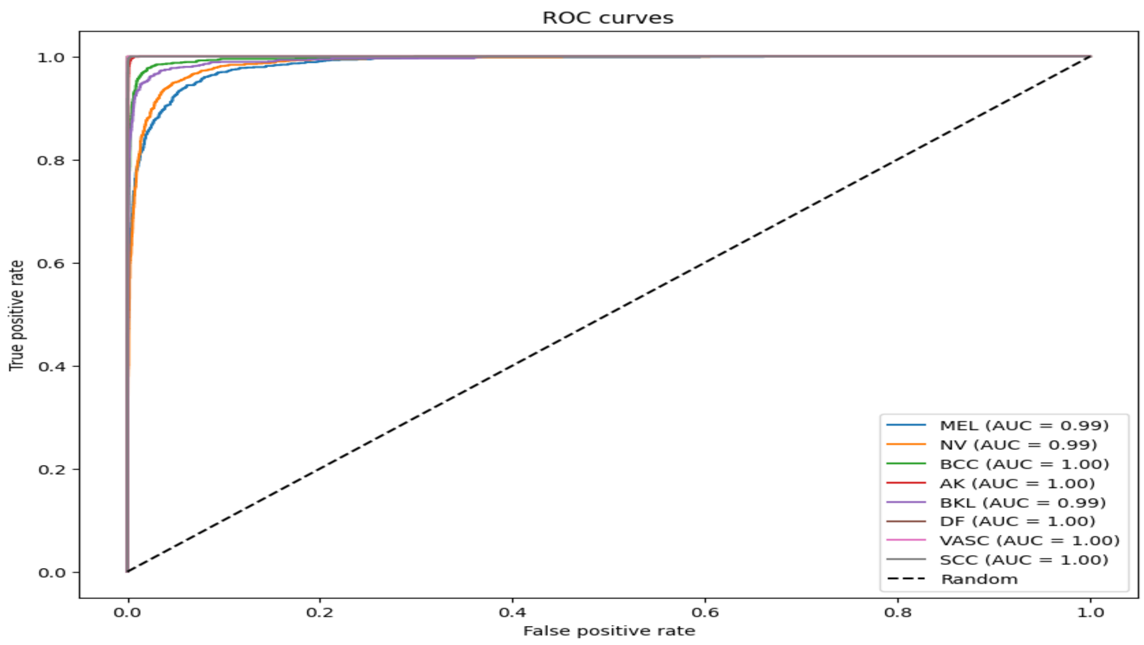


Figure 5.4.2: ROC curve after augmenting

Figure 5.4.2 above displays the true positive score at different decision rules plotted against the false positive rate score on the x-axis, representing thresholds. The orange line on the Receiver Operating Characteristics (ROC) curve illustrates the effectiveness of our algorithm, while the baseline is depicted in blue. The ROC score is computed based on the area beneath the ROC curve, and in this instance, the model's ROC score is determined to be 0.99

5.5 Images of skin lesions before and after augmentation

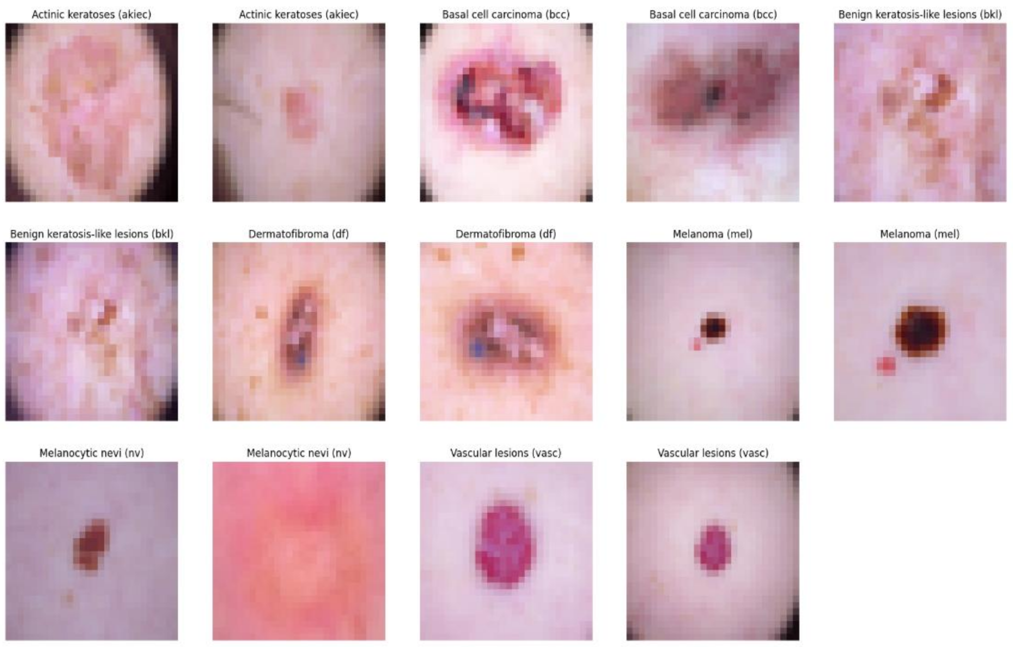

Actinic keratoses (akiec)
Actinic keratoses (akiec)
Basal cell carcinoma (bcc)
Basal cell carcinoma (bcc)
Benign keratosis-like lesions (bkl)
Benign keratosis-like lesions (bkl)
Dermatofibroma (df)
Dermatofibroma (df)
Melanoma (mel)
Melanoma (mel)
Melanocytic nevi (nv)
Melanocytic nevi (nv)
Vascular lesions (vasc)
Vascular lesions (vasc)

Figure 5.5a: Images before augmentation

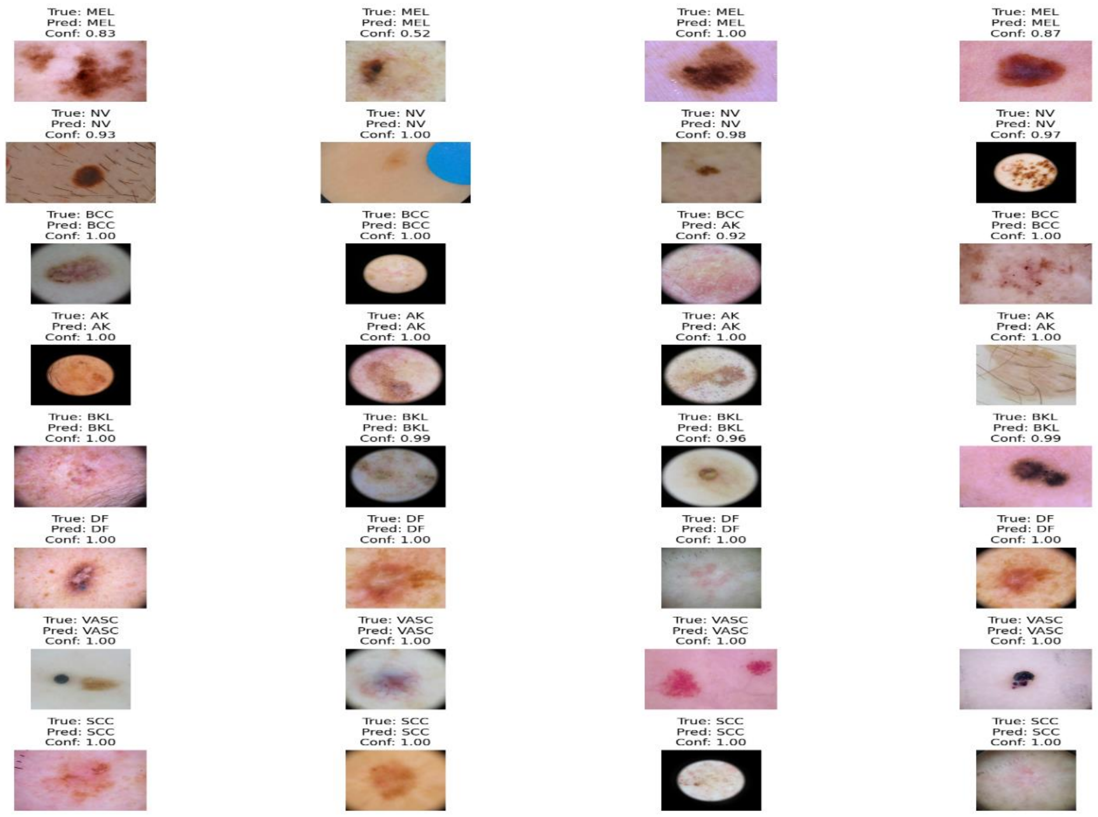
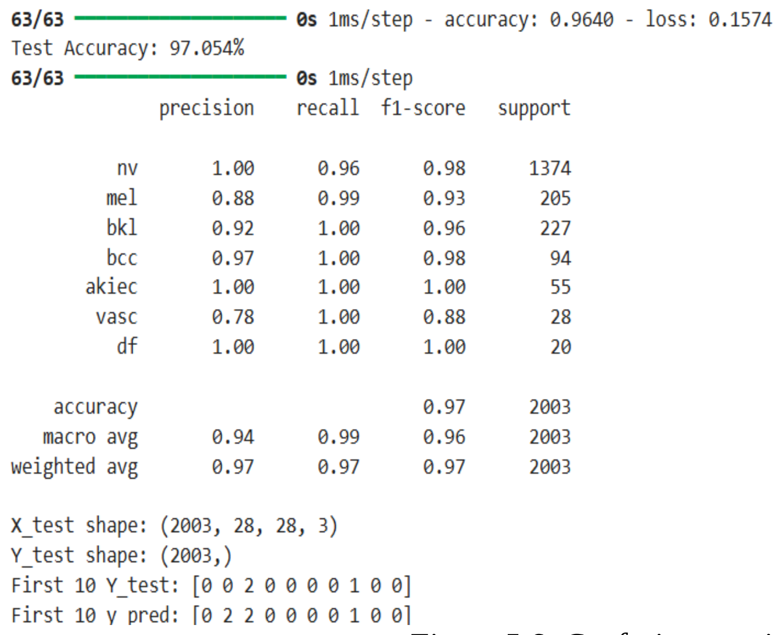

True: MEL Pred: MEL Conf: 0.83
True: MEL Pred: MEL Conf: 0.52
True: MEL Pred: MEL Conf: 1.00
True: MEL Pred: MEL Conf: 0.87
True: NV Pred: NV Conf: 0.93
True: NV Pred: NV Conf: 1.00
True: NV Pred: NV Conf: 0.98
True: NV Pred: NV Conf: 0.97
True: BCC Pred: BCC Conf: 1.00
True: BCC Pred: BCC Conf: 1.00
True: BCC Pred: AK Conf: 0.92
True: BCC Pred: BCC Conf: 1.00
True: AK Pred: AK Conf: 1.00
True: AK Pred: AK Conf: 1.00
True: AK Pred: AK Conf: 1.00
True: AK Pred: AK Conf: 1.00
True: BKL Pred: BKL Conf: 1.00
True: BKL Pred: BKL Conf: 0.99
True: BKL Pred: BKL Conf: 0.96
True: BKL Pred: BKL Conf: 0.99
True: DF Pred: DF Conf: 1.00
True: DF Pred: DF Conf: 1.00
True: DF Pred: DF Conf: 1.00
True: DF Pred: DF Conf: 1.00
True: VASC Pred: VASC Conf: 1.00
True: VASC Pred: VASC Conf: 1.00
True: VASC Pred: VASC Conf: 1.00
True: VASC Pred: VASC Conf: 1.00
True: SCC Pred: SCC Conf: 1.00
True: SCC Pred: SCC Conf: 1.00
True: SCC Pred: SCC Conf: 1.00
True: SCC Pred: SCC Conf: 1.00

Figure 5.5b: Images after augmentation

5.6 Confusion matrix of custom CNN



```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9640 - loss: 0.1574
Test Accuracy: 97.054%
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
              precision    recall  f1-score   support

          nv       1.00      0.96      0.98      1374
         mel       0.88      0.99      0.93       205
         bkl       0.92      1.00      0.96       227
         bcc       0.97      1.00      0.98        94
       akiec       1.00      1.00      1.00        55
        vasc       0.78      1.00      0.88        28
          df       1.00      1.00      1.00        20

    accuracy                           0.97      2003
   macro avg       0.94      0.99      0.96      2003
weighted avg       0.97      0.97      0.97      2003

X_test shape: (2003, 28, 28, 3)
Y_test shape: (2003,)
First 10 Y_test: [0 0 2 0 0 0 0 1 0 0]
First 10 y_pred: [0 2 2 0 0 0 0 1 0 0]
```

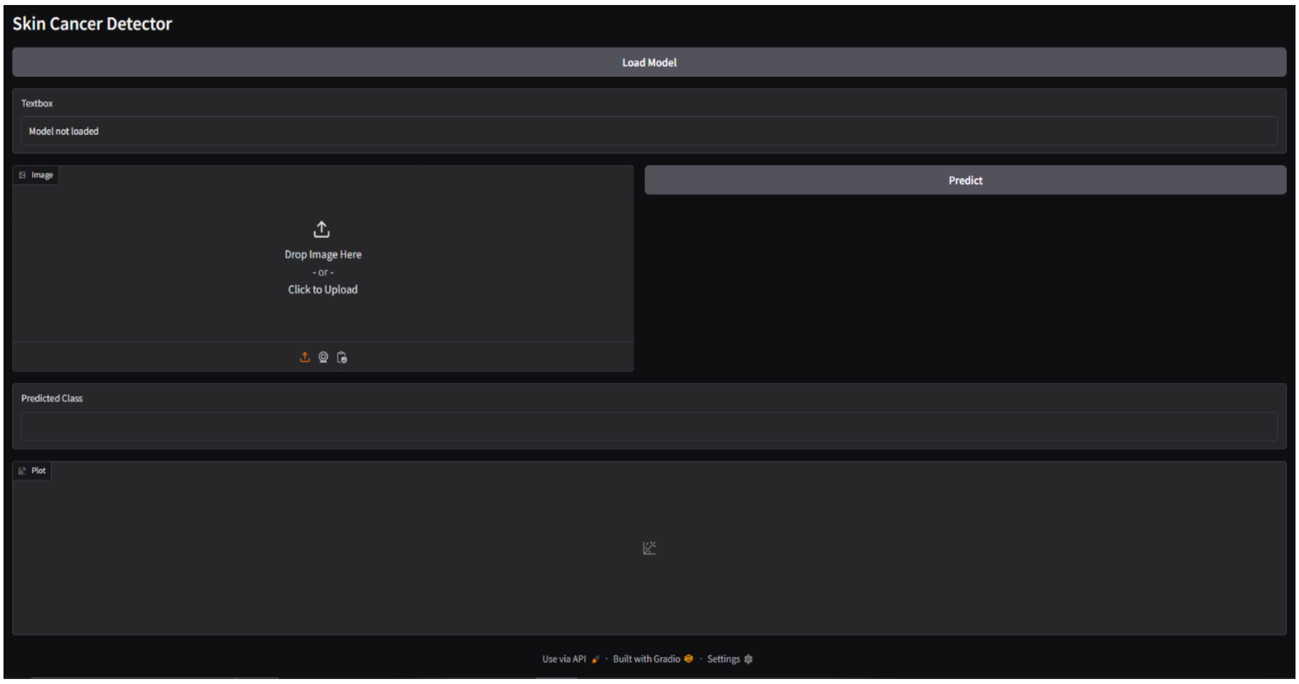
Figure 5.6: Confusion matrix

5.7 Performance analysis of different pre-trained model

Models	Accuracy
EfficientNetB7	94.17%
ResNet50	13.02%
VGG19	78.67%
MobileNet	99.68%
Custom CNN	97.05%

Table 5.7: Performance of pre-trained models

5.8 User-Interface of Gradio web application



Textbox
Model not loaded

Predicted Class

Figure 5.8: Gradio web application

Chapter-6

CONCLUSION

In this project, we successfully developed a custom CNN model for skin lesion classification, ensuring accurate predictions of various skin cancer types. The model was trained using an augmented dataset to improve generalization and performance. After training, we integrated the model into a Gradio web application, allowing users to easily upload images and receive real-time predictions with probability scores visualized through a bar chart.

The key accomplishments of this project include:

- **Custom CNN Architecture:** Designed and trained a deep learning model tailored for skin cancer classification.
- **Data Augmentation & Preprocessing:** Balanced the dataset to improve model performance.
- **Gradio Web App Implementation:** Built a user-friendly interface with options to load the model, upload images, and visualize predictions.
- **Real-time Prediction & Visualization:** Displayed classification probabilities effectively to enhance user interpretability.

This project demonstrates the potential of deep learning in medical image analysis, making early detection of skin cancer more accessible.

Chapter-7

FUTURE ENHANCEMENT

1. Optimize Deployment

- **Convert Model to TensorFlow Lite (TFLite):** Deploy on mobile devices for real-time inference.
- **Use ONNX Format:** Make the model interoperable with different frameworks like PyTorch and OpenVINO.
- **Optimize for Cloud Deployment:** Deploy on AWS, Google Cloud, or Azure for large-scale use.

2. Enhance User Experience in Gradio Web App

- **Add Heatmap Visualization:** Use Grad-CAM to highlight lesion areas contributing to the prediction.
- **Enable Multi-Image Uploads:** Allow batch processing of multiple images.
- **Provide Explanations:** Integrate LIME or SHAP to explain model decisions.

3. Integrate with Medical Databases

- **Real-Time Database Connectivity:** Store patient records securely for future reference.
- **Electronic Health Record (EHR) Integration:** Connect with hospital databases for easy access by dermatologists.

4. Expand Classification Scope

- **Detect Other Skin Conditions:** Extend the model to classify acne, psoriasis, and other common skin diseases.
- **Incorporate Multi-Class or Multi-Label Classification:** Allow multiple conditions to be detected in a single image.

5. Regulatory Compliance and Certification

- **Ensure Model Fairness:** Test on diverse datasets to avoid biases.
- **FDA/CE Certification:** Work towards making the model usable in real-world clinical applications.

Chapter-8

APPENDICES

8.1 Importing required libraries

```
# Suppress Warnings
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
warnings.simplefilter(action="ignore", category=UserWarning)
warnings.simplefilter(action="ignore",
category=RuntimeWarning)
warnings.simplefilter(action='ignore',
category=DeprecationWarning)

# Core Libraries
import os
import io
import numpy as np
import pandas as pd

# Image Processing & File Handling
from PIL import Image
from glob import glob

# Data Visualization
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning & Deep Learning
import tensorflow as tf
```

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, Flatten,
BatchNormalization, Dropout, Dense, MaxPool2D

from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping


# Scikit-learn (for data splitting & evaluation)

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report,
confusion_matrix


# Display Utilities (for Jupyter Notebooks)

from IPython.display import display

from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast_node_interactivity = "all"
```

8.2 Handling class imbalance

```
pip install imbalanced-learn

from imblearn.under_sampling import RandomUnderSampler

from imblearn.over_sampling import RandomOverSampler

import pandas as pd


# Drop 'UNK' column if it exists

if 'UNK' in df.columns:

    df = df.drop('UNK', axis=1)


# Ensure all columns are numeric

label_cols = df.columns[1:]

df[label_cols] = df[label_cols].apply(pd.to_numeric,
errors='coerce')
```

```
# Handle missing values
df.dropna(inplace=True)  # Alternative: df.fillna(0,
inplace=True)

# Convert one-hot encoded labels to a single class per row
df['label'] = df[label_cols].idxmax(axis=1)

# Ensure 'image' column exists
if 'image' not in df.columns:
    raise ValueError("The dataset must contain an 'image'
column.")

# Count class distribution
class_counts = df['label'].value_counts()
majority_class = class_counts.idxmax()
majority_class_count = class_counts.max()

# Step-1: Undersample the majority class
undersample_ratio = 0.5  # Adjust as needed
undersample_strategy = {majority_class: int(undersample_ratio
* majority_class_count)}

rus =
RandomUnderSampler(sampling_strategy=undersample_strategy,
random_state=42)
X_under, Y_under = rus.fit_resample(df[['image']],
df['label'])

# Create a new DataFrame after undersampling
df_under = pd.DataFrame(X_under, columns=['image'])
df_under['label'] = Y_under

# Step-2: Oversample minority classes
```

```
class_counts_under = df_under['label'].value_counts()
new_majority_class_count = class_counts_under.max()


# Define oversampling strategy
sampling_strategy = {cls: new_majority_class_count for cls in
class_counts_under.index}
ros = RandomOverSampler(sampling_strategy=sampling_strategy,
random_state=42)


X_resampled, Y_resampled =
ros.fit_resample(df_under[['image']], df_under['label'])


# Convert back to one-hot encoding
df_resampled = pd.DataFrame(X_resampled, columns=['image'])
one_hot_labels = pd.get_dummies(Y_resampled)


# Ensure all original labels exist in one-hot encoding
for col in label_cols:
    if col not in one_hot_labels.columns:
        one_hot_labels[col] = 0


# Merge back into final dataset
df_resampled = pd.concat([df_resampled, one_hot_labels],
axis=1)


# Save the balanced dataset
df_resampled.to_csv('Balanced_Dataset.csv', index=False)
print('Balanced dataset saved successfully with full class
balance.')
```

8.3 Splitting the data into training and testing

```
def prepare_for_train_test(X, Y, test_size=0.2):
    # Split the dataset
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size, random_state=1)

    # Normalize pixel values (scale between 0 and 1)
    X_train = X_train.astype("float32") / 255.0
    X_test = X_test.astype("float32") / 255.0

    # Convert labels to 1D if needed
    Y_train = Y_train.flatten()
    Y_test = Y_test.flatten()

    return X_train, X_test, Y_train, Y_test
```

8.4 Creating custom CNN model

```
def create_model():
    model = Sequential()
    model.add(Conv2D(16, kernel_size=(3,3), input_shape=(28,
28, 3), activation='relu', padding='same'))
    model.add(MaxPool2D(pool_size=(2,2)))  # Removed invalid
padding

    model.add(Conv2D(32, kernel_size=(3,3), activation='relu',
padding='same'))
    model.add(MaxPool2D(pool_size=(2,2)))  # Removed invalid
padding

    model.add(Conv2D(64, kernel_size=(3,3), activation='relu',
padding='same'))
    model.add(MaxPool2D(pool_size=(2,2)))  # Removed invalid
padding
```

```
    model.add(Conv2D(128, kernel_size=(3,3),
activation='relu', padding='same'))

    model.add(MaxPool2D(pool_size=(2,2)))  # Removed invalid
padding


    model.add(Flatten())

    model.add(Dense(64, activation='relu'))

    model.add(Dense(32, activation='relu'))

    model.add(Dense(7, activation='softmax'))  # Assuming 7
classes


    optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)


    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])


    model.summary()  # Prints the model structure

    return model
```

8.5 Training the model

```
def train_model(model, X_train, Y_train, EPOCHS=25):

    early_stop = EarlyStopping(monitor='val_loss',
patience=10, verbose=1, mode='min')


    reduce_lr = ReduceLROnPlateau(monitor='val_loss',
factor=0.1, patience=3, verbose=1, mode='min')


    history = model.fit(X_train,
                        Y_train,
                        validation_split=0.2,  # Ensure you
haven't pre-split data
                        batch_size=64,
                        epochs=EPOCHS,
                        shuffle=True,  # Ensures better
training stability
                        callbacks=[reduce_lr, early_stop])
    return history
```

8.6 Merging HAM10000_images_part_1 & HAM10000_images_part_2

```
import os
from glob import glob


# Define the correct base directory (Kaggle or local)
base_skin_dir = "/kaggle/input/skin-cancer-mnist-ham10000"


# Merging images from both folders HAM10000_images_part_1 &
HAM10000_images_part_2
image_paths = glob(os.path.join(base_skin_dir,
"HAM10000_images_part_1", "*.jpg")) + \
              glob(os.path.join(base_skin_dir,
"HAM10000_images_part_2", "*.jpg"))


# Remove duplicates (some images exist in both folders)
imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]:
x for x in image_paths}


# Check if images are loaded correctly
print(f"Total unique images found: {len(imageid_path_dict)}")
print("Sample paths:", list(imageid_path_dict.items())[:5])  #
Show first 5 image mappings


# Mapping lesion types to their full names
lesion_type_dict = {
    'nv': 'Melanocytic nevi (nv)',
    'mel': 'Melanoma (mel)',
    'bkl': 'Benign keratosis-like lesions (bkl)',
    'bcc': 'Basal cell carcinoma (bcc)',
    'akiec': 'Actinic keratoses (akiec)',
    'vasc': 'Vascular lesions (vasc)',
    'df': 'Dermatofibroma (df)'
}

# Mapping numerical labels to lesion types
label_mapping = {
    0: 'nv',
    1: 'mel',
    2: 'bkl',
    3: 'bcc',
    4: 'akiec',
    5: 'vasc',
    6: 'df'
}

# Reverse mapping lesion type to numerical label
reverse_label_mapping = {v: k for k, v in
label_mapping.items()}

# Print mappings for verification
print(label_mapping)
print(reverse_label_mapping)
print(lesion_type_dict)
```

8.7 Adding cell type and image paths

```
# Adding cell_type and image_path columns
data['cell_type'] = data['dx'].map(lesion_type_dict.get)
data['path'] = data['image_id'].map(imageid_path_dict.get)

# Check if there are any missing paths
missing_paths = data[data['path'].isna()]
```

```
print(f"Missing image paths: {len(missing_paths)}")


# Check if the mappings look correct
print(data[['image_id', 'path', 'cell_type']].head())
```

8.8 Mapping

```
# Map dx to numerical labels
data['label'] = data['dx'].map(reverse_label_mapping.get)


# Handle any missing values in 'label' (optional, depending on
dataset)
data = data.dropna(subset=['label'])


# Convert 'label' to int (if needed)
data['label'] = data['label'].astype(int)


# Sort by label and reset index
data = data.sort_values('label').reset_index(drop=True)
```

8.9 Original data

```
# Converting image pixel column into required format
X_orig = data['image_pixel'].to_numpy()
X_orig = np.stack(X_orig, axis=0)
Y_orig = np.array(data.iloc[:, -1:])
print(X_orig.shape)
print(Y_orig.shape)
```

8.10 Augmented data

```
# Converting image pixel columnm into required format
X_aug = final_data['image_pixel'].to_numpy()
X_aug = np.stack(X_aug, axis=0)
Y_aug = np.array(final_data.iloc[:, -1:])
print(X_aug.shape)
print(Y_aug.shape)
```

8.11 Accuracy, classification report and prediction

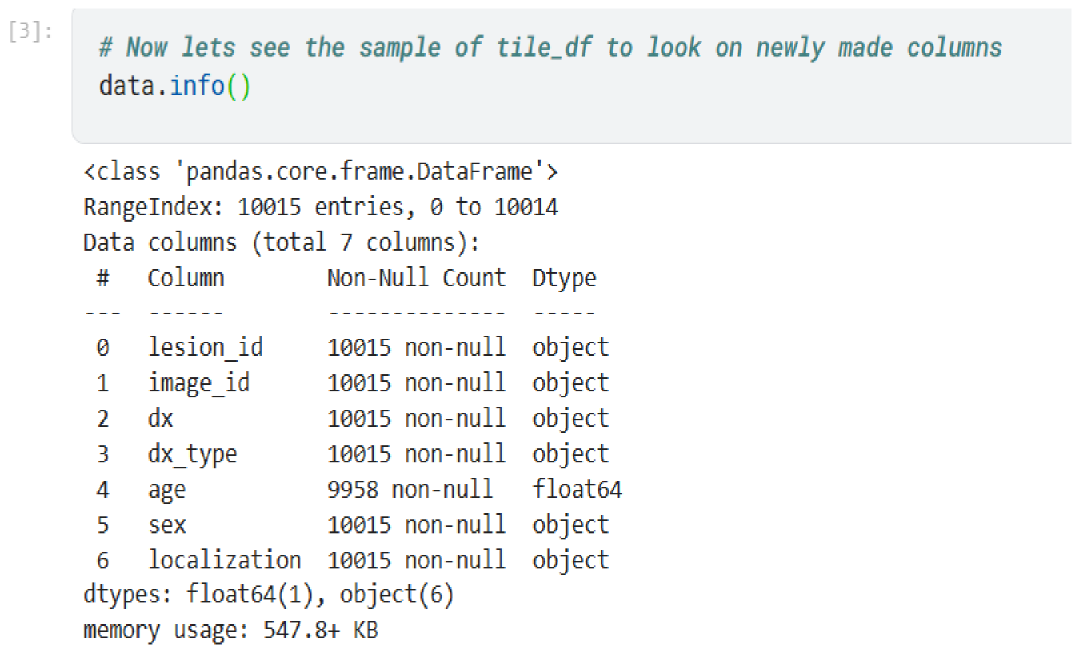
```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import classification_report


def test_model(model, X_test, Y_test):

    # Evaluate model accuracy

    model_acc = model.evaluate(X_test, Y_test, verbose=1)[1]

    print("Test Accuracy: {:.3f}%".format(model_acc * 100))


    # Ensure Y_test is 1D

    y_true = np.array(Y_test).flatten()


    # Get model predictions

    y_pred = model.predict(X_test)

    y_pred = np.argmax(y_pred, axis=1)


    # Print classification report

    clr = classification_report(y_true, y_pred,
target_names=label_mapping.values())

    print(clr)


    # Debugging prints

    print(f"X_test shape: {X_test.shape}")

    print(f"Y_test shape: {Y_test.shape}")

    print(f"First 10 Y_test: {y_true[:10]}")

    print(f"First 10 y_pred: {y_pred[:10]}")


    # Display 15 sample images

    plt.figure(figsize=(22, 12))

    for i in range(min(15, len(X_test))):  # Prevent errors if fewer
than 15 samples

        plt.subplot(3, 5, i + 1)
        plt.imshow(X_test[i])

        plt.title(f"True: {label_mapping[y_true[i]]} | Pred:
{label_mapping[y_pred[i]]}")

        plt.axis("off")


    # Ensure plot displays

    plt.show(block=True)


    # Save image in case display fails

    plt.savefig("output.png")

    print("Plot saved as 'output.png'. Check if it exists.")


test_model(model, X_test_orig, Y_test_orig)
```

8.12 Gradio

```
pip install gradio

import gradio as gr

import tensorflow as tf

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt


# Global variable to store the model

model = None

class_labels = ['nv', 'mel', 'bkl', 'bcc', 'akiec', 'vasc',
'df']

def load_model():

    global model

    model =
tf.keras.models.load_model("/content/drive/MyDrive/skin_cancer
_prediction.h5")

    return "Model Loaded Successfully!"


def predict(image):
    global model
    if model is None:
        return "Please load the model first!", None

    image = image.resize((28, 28))  # Resize to model input
size
    image = np.array(image) / 255.0  # Normalize
    image = np.expand_dims(image, axis=0)  # Add batch
dimension

    predictions = model.predict(image)[0]  # Get predictions
    predicted_class = class_labels[np.argmax(predictions)]  #
Get class label

    # Create a bar chart for probabilities
    fig, ax = plt.subplots()
    ax.bar(class_labels, predictions, color='skyblue')
    ax.set_xlabel("Classes")
    ax.set_ylabel("Probability")
    ax.set_title("Prediction Probabilities")

    return predicted_class, fig

with gr.Blocks() as app:
    gr.Markdown("# Skin Cancer Detector")

    load_button = gr.Button("Load Model")
    model_status = gr.Textbox("Model not loaded",
interactive=False)

    load_button.click(load_model, outputs=model_status)
    with gr.Row():
        image_input = gr.Image(type="pil")
        predict_button = gr.Button("Predict")

    output_label = gr.Textbox(label="Predicted Class")
    output_plot = gr.Plot()

    predict_button.click(predict, inputs=image_input,
outputs=[output_label, output_plot])

app.launch()
```

8.13 Screenshots

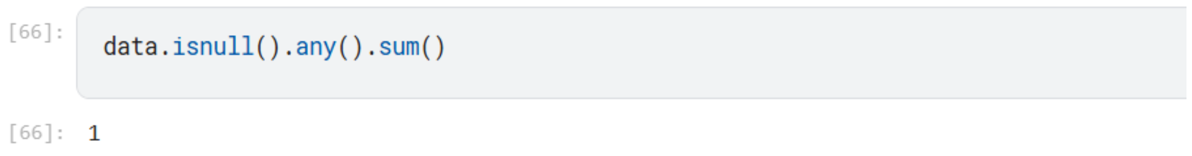
8.13.1 Loading the dataset



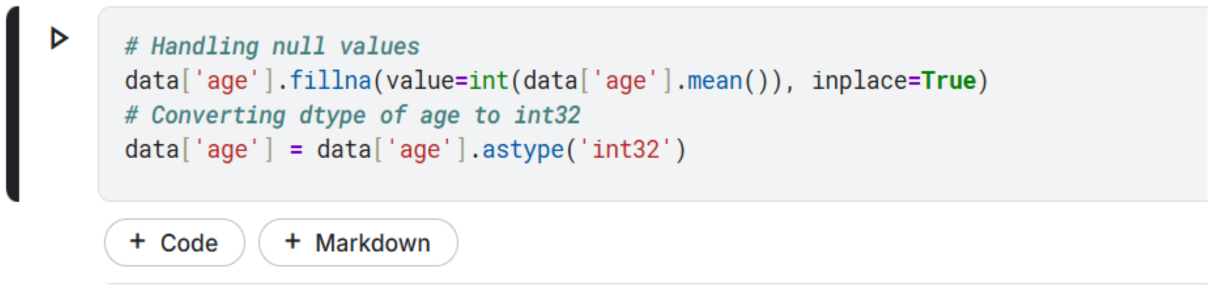
```
[3]: # Now lets see the sample of tile_df to look on newly made columns
     data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10015 entries, 0 to 10014
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   lesion_id     10015 non-null  object 
 1   image_id      10015 non-null  object 
 2   dx            10015 non-null  object 
 3   dx_type       10015 non-null  object 
 4   age           9958 non-null   float64
 5   sex           10015 non-null  object 
 6   localization  10015 non-null  object 
dtypes: float64(1), object(6)
memory usage: 547.8+ KB
```

8.13.2 Handling missing values



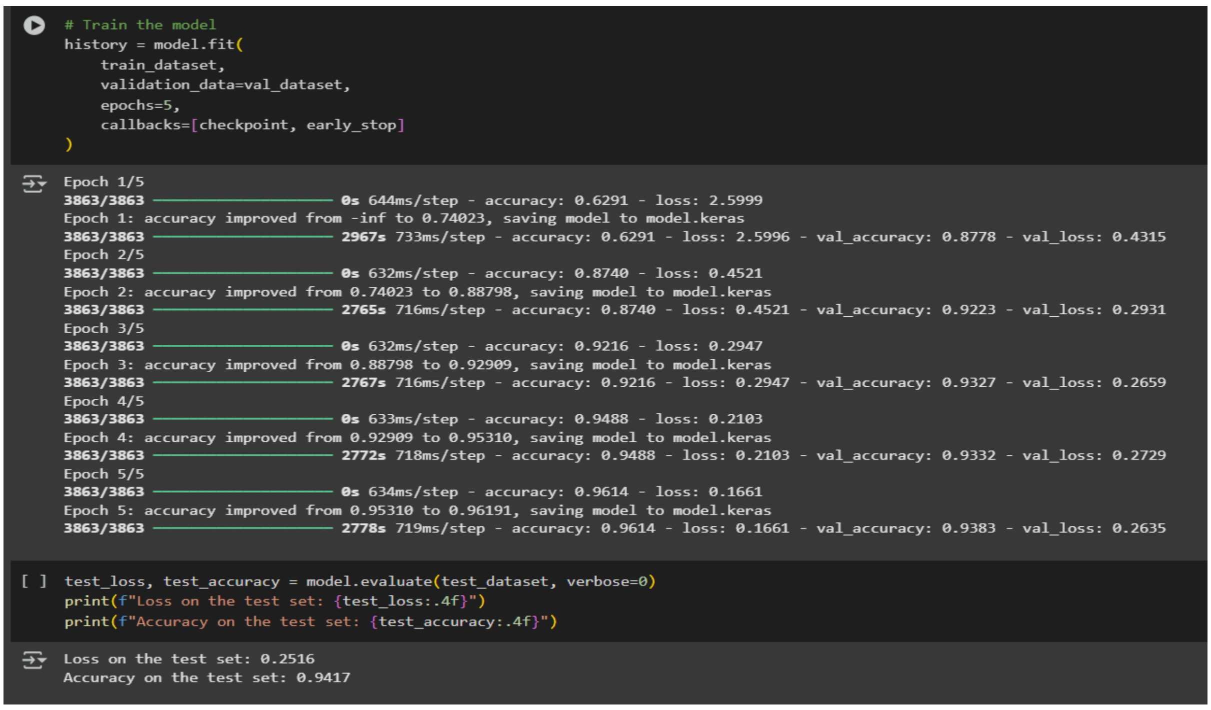
```
[66]: data.isnull().any().sum()

[66]: 1
```



```
# Handling null values
data['age'].fillna(value=int(data['age'].mean()), inplace=True)
# Converting dtype of age to int32
data['age'] = data['age'].astype('int32')
```

8.14 Comparing with different pre-trained models

8.14.1 EfficientNetB7



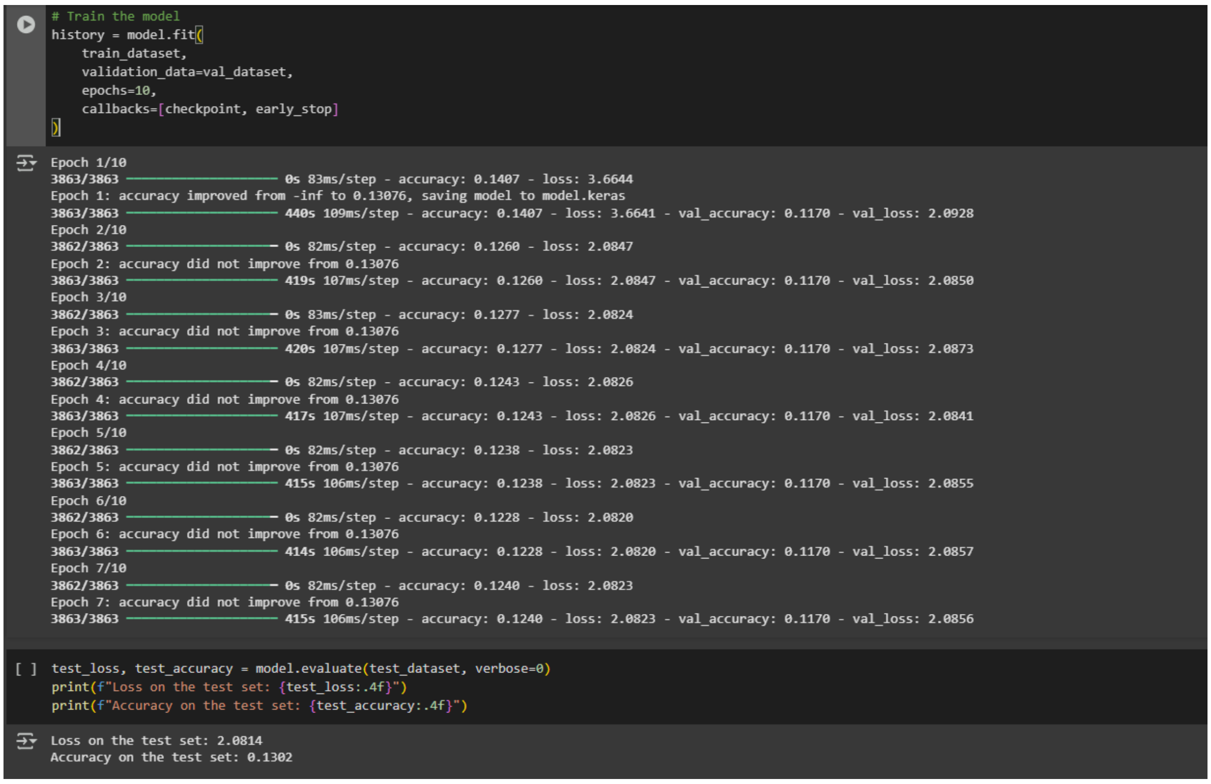
```
# Train the model
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=5,
    callbacks=[checkpoint, early_stop]
)
```

```
Epoch 1/5
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 644ms/step - accuracy: 0.6291 - loss: 2.5999
Epoch 1: accuracy improved from -inf to 0.74023, saving model to model.keras
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 2967s 733ms/step - accuracy: 0.6291 - loss: 2.5996 - val_accuracy: 0.8778 - val_loss: 0.4315
Epoch 2/5
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 632ms/step - accuracy: 0.8740 - loss: 0.4521
Epoch 2: accuracy improved from 0.74023 to 0.88798, saving model to model.keras
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 2765s 716ms/step - accuracy: 0.8740 - loss: 0.4521 - val_accuracy: 0.9223 - val_loss: 0.2931
Epoch 3/5
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 632ms/step - accuracy: 0.9216 - loss: 0.2947
Epoch 3: accuracy improved from 0.88798 to 0.92909, saving model to model.keras
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 2767s 716ms/step - accuracy: 0.9216 - loss: 0.2947 - val_accuracy: 0.9327 - val_loss: 0.2659
Epoch 4/5
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 633ms/step - accuracy: 0.9488 - loss: 0.2103
Epoch 4: accuracy improved from 0.92909 to 0.95310, saving model to model.keras
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 2772s 718ms/step - accuracy: 0.9488 - loss: 0.2103 - val_accuracy: 0.9332 - val_loss: 0.2729
Epoch 5/5
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 634ms/step - accuracy: 0.9614 - loss: 0.1661
Epoch 5: accuracy improved from 0.95310 to 0.96191, saving model to model.keras
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 2778s 719ms/step - accuracy: 0.9614 - loss: 0.1661 - val_accuracy: 0.9383 - val_loss: 0.2635
```

```
[ ] test_loss, test_accuracy = model.evaluate(test_dataset, verbose=0)
    print(f"Loss on the test set: {test_loss:.4f}")
    print(f"Accuracy on the test set: {test_accuracy:.4f}")
```

```
Loss on the test set: 0.2516
Accuracy on the test set: 0.9417
```

8.14.2 ResNet50



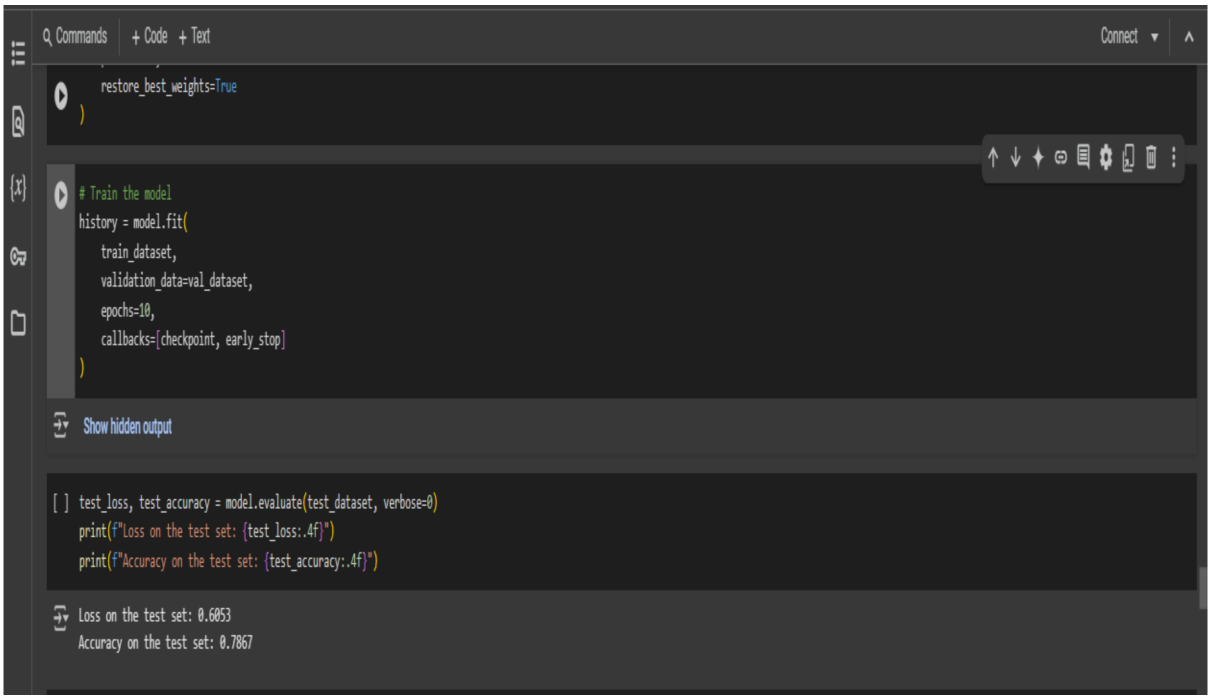
```
# Train the model
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=10,
    callbacks=[checkpoint, early_stop]
)
```

```
Epoch 1/10
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.1407 - loss: 3.6644
Epoch 1: accuracy improved from -inf to 0.13076, saving model to model.keras
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 440s 109ms/step - accuracy: 0.1407 - loss: 3.6641 - val_accuracy: 0.1170 - val_loss: 2.0928
Epoch 2/10
3862/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.1260 - loss: 2.0847
Epoch 2: accuracy did not improve from 0.13076
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 419s 107ms/step - accuracy: 0.1260 - loss: 2.0847 - val_accuracy: 0.1170 - val_loss: 2.0850
Epoch 3/10
3862/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.1277 - loss: 2.0824
Epoch 3: accuracy did not improve from 0.13076
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 420s 107ms/step - accuracy: 0.1277 - loss: 2.0824 - val_accuracy: 0.1170 - val_loss: 2.0873
Epoch 4/10
3862/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.1243 - loss: 2.0826
Epoch 4: accuracy did not improve from 0.13076
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 417s 107ms/step - accuracy: 0.1243 - loss: 2.0826 - val_accuracy: 0.1170 - val_loss: 2.0841
Epoch 5/10
3862/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.1238 - loss: 2.0823
Epoch 5: accuracy did not improve from 0.13076
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 415s 106ms/step - accuracy: 0.1238 - loss: 2.0823 - val_accuracy: 0.1170 - val_loss: 2.0855
Epoch 6/10
3862/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.1228 - loss: 2.0820
Epoch 6: accuracy did not improve from 0.13076
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 414s 106ms/step - accuracy: 0.1228 - loss: 2.0820 - val_accuracy: 0.1170 - val_loss: 2.0857
Epoch 7/10
3862/3863 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.1240 - loss: 2.0823
Epoch 7: accuracy did not improve from 0.13076
3863/3863 ━━━━━━━━━━━━━━━━━━━━ 415s 106ms/step - accuracy: 0.1240 - loss: 2.0823 - val_accuracy: 0.1170 - val_loss: 2.0856
```

```
[ ] test_loss, test_accuracy = model.evaluate(test_dataset, verbose=0)
    print(f"Loss on the test set: {test_loss:.4f}")
    print(f"Accuracy on the test set: {test_accuracy:.4f}")
```

```
Loss on the test set: 2.0814
Accuracy on the test set: 0.1302
```

8.14.3 VGG19



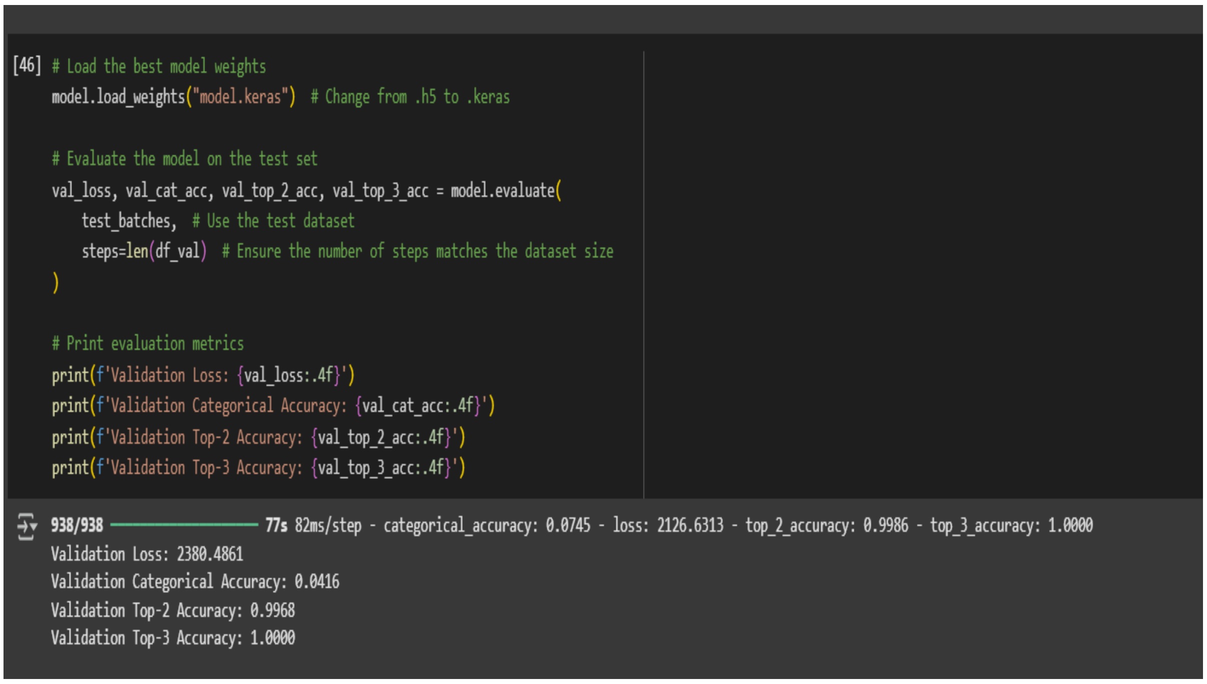
```
    restore_best_weights=True
)
```

```
# Train the model
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=10,
    callbacks=[checkpoint, early_stop]
)
```

```
[ ] test_loss, test_accuracy = model.evaluate(test_dataset, verbose=0)
    print(f"Loss on the test set: {test_loss:.4f}")
    print(f"Accuracy on the test set: {test_accuracy:.4f}")
```

```
Loss on the test set: 0.6053
Accuracy on the test set: 0.7867
```

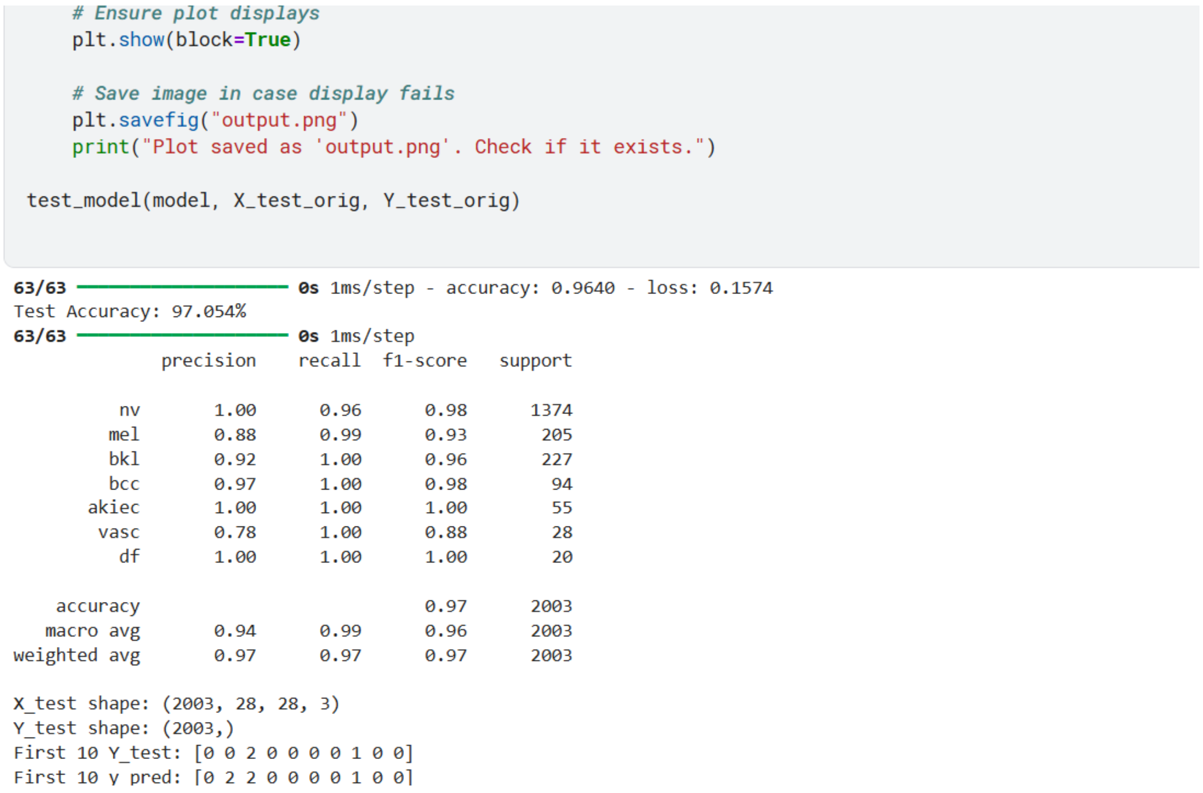
8.14.4 MobileNetV2



```
[46] # Load the best model weights
     model.load_weights("model.keras")  # Change from .h5 to .keras

     # Evaluate the model on the test set
     val_loss, val_cat_acc, val_top_2_acc, val_top_3_acc = model.evaluate(
         test_batches,  # Use the test dataset
         steps=len(df_val)  # Ensure the number of steps matches the dataset size
     )

     # Print evaluation metrics
     print(f'Validation Loss: {val_loss:.4f}')
     print(f'Validation Categorical Accuracy: {val_cat_acc:.4f}')
     print(f'Validation Top-2 Accuracy: {val_top_2_acc:.4f}')
     print(f'Validation Top-3 Accuracy: {val_top_3_acc:.4f}')
```

```
938/938 ━━━━━━━━━━━━━━━━━━━━ 77s 82ms/step - categorical_accuracy: 0.0745 - loss: 2126.6313 - top_2_accuracy: 0.9986 - top_3_accuracy: 1.0000
Validation Loss: 2380.4861
Validation Categorical Accuracy: 0.0416
Validation Top-2 Accuracy: 0.9968
Validation Top-3 Accuracy: 1.0000
```

8.14.5 Custom CNN



```
    # Ensure plot displays
    plt.show(block=True)

    # Save image in case display fails
    plt.savefig("output.png")
    print("Plot saved as 'output.png'. Check if it exists.")

test_model(model, X_test_orig, Y_test_orig)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9640 - loss: 0.1574
Test Accuracy: 97.054%
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
              precision    recall  f1-score   support

          nv       1.00      0.96      0.98      1374
         mel       0.88      0.99      0.93       205
         bkl       0.92      1.00      0.96       227
         bcc       0.97      1.00      0.98        94
       akiec       1.00      1.00      1.00        55
        vasc       0.78      1.00      0.88        28
          df       1.00      1.00      1.00        20

    accuracy                           0.97      2003
   macro avg       0.94      0.99      0.96      2003
weighted avg       0.97      0.97      0.97      2003

X_test shape: (2003, 28, 28, 3)
Y_test shape: (2003,)
First 10 Y_test: [0 0 2 0 0 0 0 1 0 0]
First 10 y_pred: [0 2 2 0 0 0 0 1 0 0]
```

Chapter-9

REFERENCES

Base Paper:

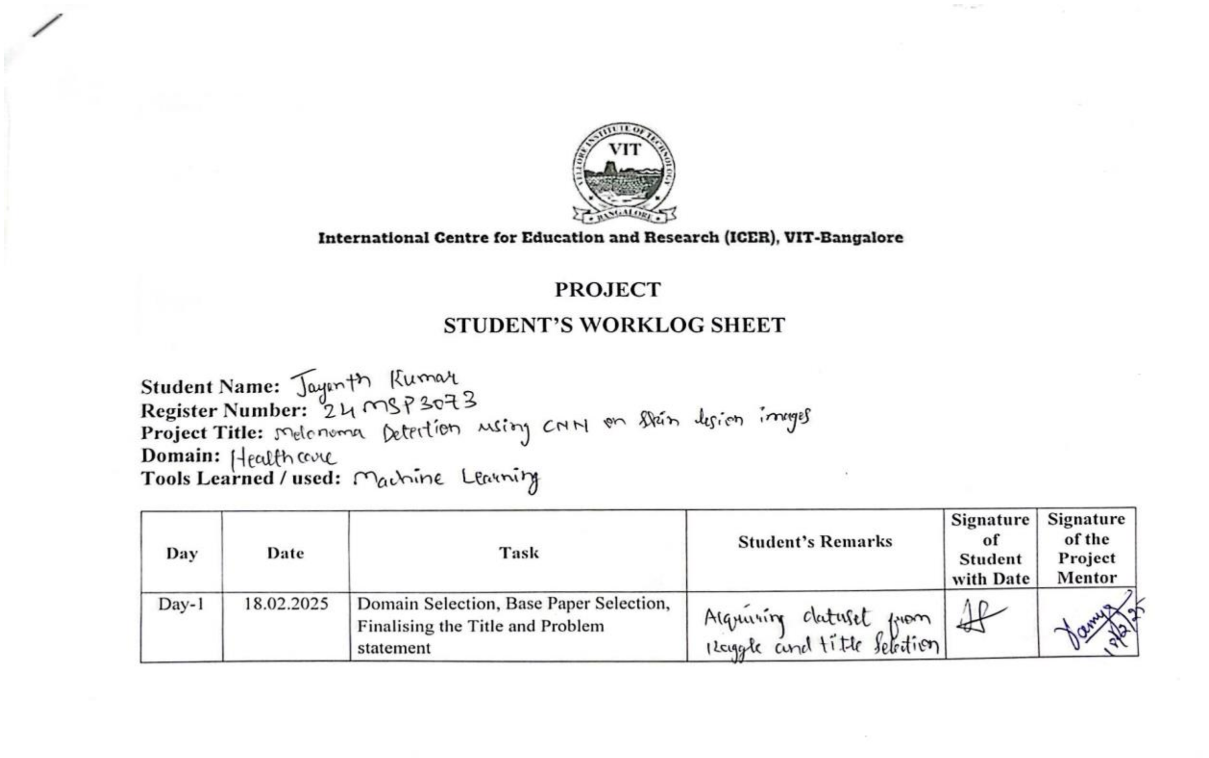
- Nils Gessert, Maximilian Nielsen, Mohsin Shaikh, Rene Wener, Alexander Schlaefer (2020), “Skin Lesion classification using ensembles of multi-resolution EfficientNets with meta data” in Elsevier
- DOI: https://doi.org/10.1016/j.mex.2020.100864

Related Papers:

- Bhuvaneshwari Shetty, Roshan Fernandes, Anisha P. Rodrigues, Rajeswari Chengoden, Sweta Bhattacharya & Kuruva Lakshmanna (2022), “Skin lesion classification of dermoscopic images using machine learning and convolutional neural network”, Sci Rep 12, 18134.
- DOI: https://doi.org/10.1038/s41598-022-22644-9
- Thwin, S. M., & Park, H.-S. (2024). Skin Lesion Classification Using a Deep Ensemble Model. Applied Sciences, 14(13), 5599.
- DOI: https://doi.org/10.3390/app14135599
- A. Afroz, R. Zia, A. O. Garcia, M. U. Khan, U. Jilani and K. M. Ahmed (2022) "Skin lesion classification using machine learning approach: A survey," 2022 Global Conference on Wireless and Optical Technologies (GCWOT), Malaga, Spain, 2022, pp. 1-8
- DOI: 10.1109/GCWOT53057.2022.9772915.

Chapter-10

WORKLOG SHEET



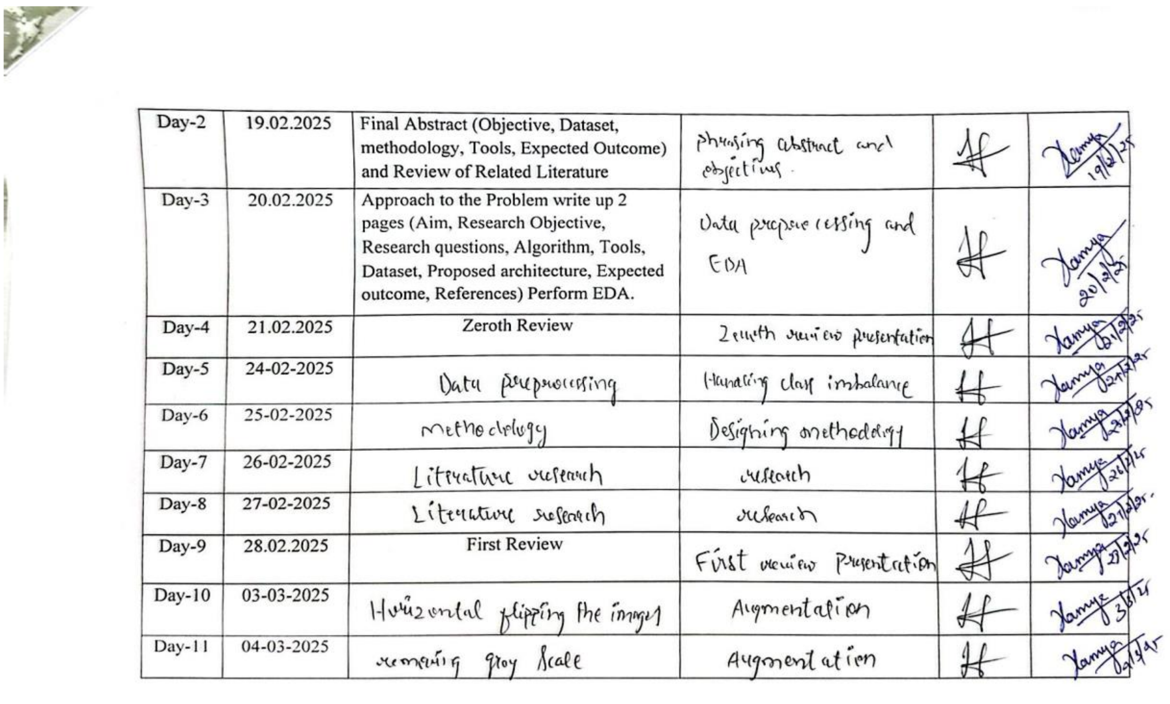
International Centre for Education and Research (ICER), VIT-Bangalore

PROJECT

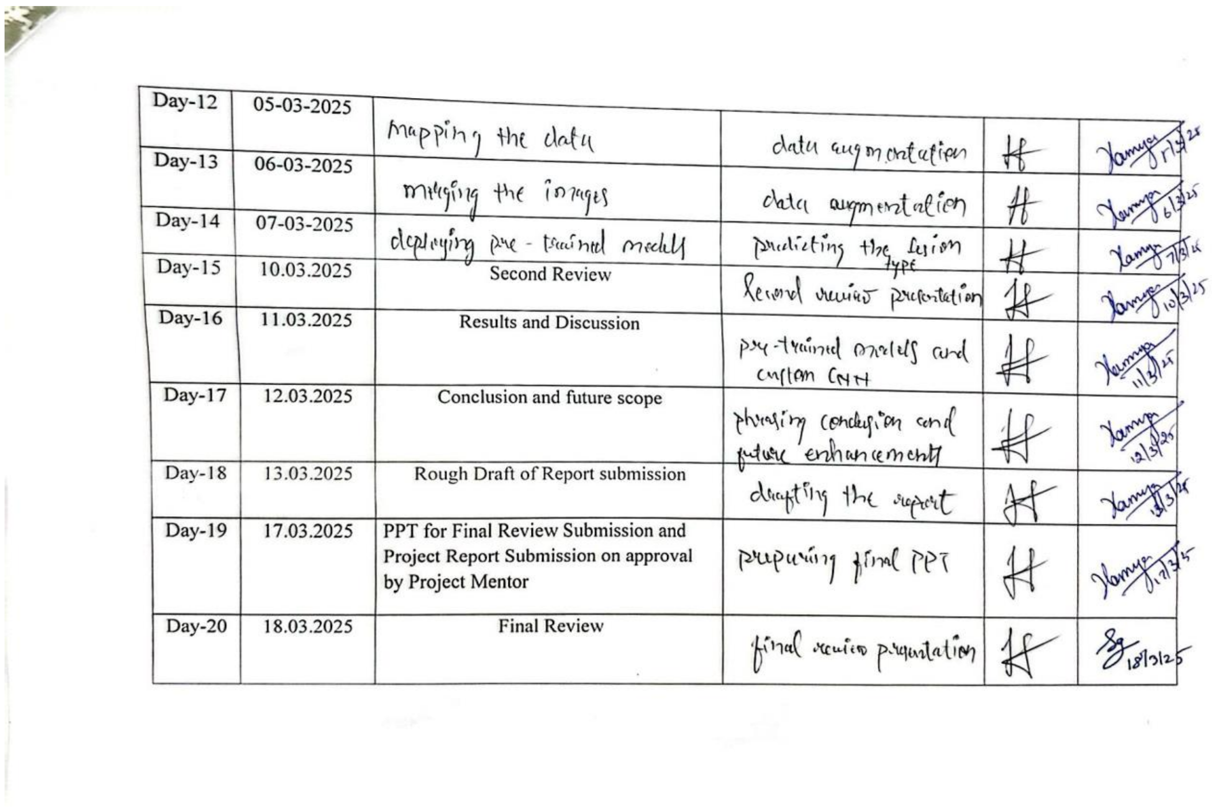
STUDENT'S WORKLOG SHEET

Student Name: Jayanth Kumar
Register Number: 24MSP3073
Project Title: Melanoma Detection using CNN on Skin lesion images
Domain: Healthcare
Tools Learned / used: Machine Learning

Day	Date	Task	Student's Remarks	Signature of Student with Date	Signature of the Project Mentor
Day-1	18.02.2025	Domain Selection, Base Paper Selection, Finalising the Title and Problem statement	Acquiring dataset from kaggle and title Selection	[illegible]	[illegible] 19/2/25



Day-2	19.02.2025	Final Abstract (Objective, Dataset, methodology, Tools, Expected Outcome) and Review of Related Literature	Phrasing abstract and objectives.	[illegible]	[illegible] 19/2/25
Day-3	20.02.2025	Approach to the Problem write up 2 pages (Aim, Research Objective, Research questions, Algorithm, Tools, Dataset, Proposed architecture, Expected outcome, References) Perform EDA.	Data preprocessing and EDA	[illegible]	[illegible] 20/2/25
Day-4	21.02.2025	Zeroth Review	Zeroth review presentation	[illegible]	[illegible] 21/2/25
Day-5	24-02-2025	Data preprocessing	Handling class imbalance	[illegible]	[illegible] 24/2/25
Day-6	25-02-2025	methodology	Designing methodology	[illegible]	[illegible] 25/2/25
Day-7	26-02-2025	Literature research	research	[illegible]	[illegible] 26/2/25
Day-8	27-02-2025	Literature research	research	[illegible]	[illegible] 27/2/25.
Day-9	28.02.2025	First Review	First review Presentation	[illegible]	[illegible] 28/2/25
Day-10	03-03-2025	Horizontal flipping the images	Augmentation	[illegible]	[illegible] 3/3/25
Day-11	04-03-2025	removing gray Scale	Augmentation	[illegible]	[illegible] 4/3/25



Day-12	05-03-2025	Mapping the data	data augmentation	[illegible]	[illegible]
Day-13	06-03-2025	merging the images	data augmentation	[illegible]	[illegible]
Day-14	07-03-2025	deploying pre-trained models	predicting the lesion type	[illegible]	[illegible]
Day-15	10.03.2025	Second Review	second review presentation	[illegible]	[illegible]
Day-16	11.03.2025	Results and Discussion	pre-trained models and custom CNN	[illegible]	[illegible]
Day-17	12.03.2025	Conclusion and future scope	phrasing conclusion and future enhancements	[illegible]	[illegible]
Day-18	13.03.2025	Rough Draft of Report submission	drafting the report	[illegible]	[illegible]
Day-19	17.03.2025	PPT for Final Review Submission and Project Report Submission on approval by Project Mentor	preparing final PPT	[illegible]	[illegible]
Day-20	18.03.2025	Final Review	final review presentation	[illegible]	[illegible] 18/3/25