MAR BASELIOS INSTITUTE OF TECHNOLOGY AND SCIENCE
Nellimattom , Kothamangalam
(Affiliated to APJ Abdul Kalam Technological University, TVM)

Department of Computer Applications

Mini Project Report

SKIN CANCER DETECTION USING CNN

Done by

MEENAKSHI A M

Reg No: MBI23MCA-2025

Under the guidance of
PROF. SUMI K YELDHO

2023-2025

CERTIFICATE

SKIN CANCER DETECTION USING CNN

Certified that this is the bonafide record of project work done by

MEENAKSHI A M

Reg No: MBI23MCA-2025

During the academic year 2023-2025, in partial fulfillment of requirements for award of the degree,

Master of Computer Applications
of
APJ Abdul Kalam Technological University
Thiruvananthapuram

Faculty Guide
PROF. SUMI K YELDHO

Head of the Department
Prof. RESHMA S

Project Coordinator
Prof. MERIN JOY M

Internal Examiner

Table of Contents

ACKNOWLEDGEMENT

First and foremost, I thank God Almighty for his divine grace and blessings in making all this possible. May he continue to lead me in the years to come. No words can express my humble gratitude to my beloved parents who have been guiding me in all walks of my journey.

I am also grateful to Prof. Reshma S, Head of Computer Applications Department, Prof. Merin Joy M, Project Coordinator for their valuable guidance and constant supervision as well as for providing necessary information regarding the project & also for their support.

I would like to express my special gratitude and thanks to my Project Guide Prof. Sumi K Yeldho, Assistant Professor, Department of Computer Applications for giving me such attentionand time.

I profusely thank other Professors in the department and all other staffs of MBITS, for their guidance and inspirations throughout my course of study. My thanks and appreciations also goes to my friends and people who have willingly helped me out with their abilities.

ABSTRACT

Skin cancer is one of the most dangerous forms of cancer. Skin cancer is caused by un-repaired deoxyribonucleic acid (DNA) in skin cells, which generate genetic defects or mutations on the skin. Skin cancer tends to gradually spread over other body parts, so it is more curable in initial stages, which is why it is best detected at early stages. The increasing rate of skin cancer cases, high mortality rate, and expensive medical treatment require that its symptoms be diagnosed early. Lesion parameters such as symmetry, color, size, shape, etc. are used to detect skin cancer and to distinguish benign skin cancer from melanoma.

A Convolutional Neural Network (CNN) is a specialized neural network architecture designed for processing grid-like data, such as images. The convolutional layer is the core component of CNNs, responsible for the majority of the network's computational work. CNNs are particularly effective for detecting and classifying skin lesions, as our inputs consist of image files. While training deep networks can be challenging due to issues like the vanishing gradient problem, our architecture is designed to address these challenges through careful layer configuration. The model utilizes convolutional layers followed by batch normalization and max pooling to extract features from the images. This allows the network to learn essential patterns for accurate classification of skin lesions. The dataset used for training and testing includes a variety of labeled images of skin lesions, enabling the model to learn and enhance its classification performance.

REFERENCES:

- Malo, Dipu Chandra, et al. "Skin cancer detection using convolutional neural network." *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2022.
- Sharma, Deepti, and Swati Srivastava. "Automatically detection of skin cancer by classification of neural network." *International Journal of Engineering and Technical Research* 4.1 (2016): 15-18.

1. INTRODUCTION

Skin cancer is a significant health concern, and early detection is vital for effective treatment. Our project addresses this issue by developing a convolutional neural network (CNN) model to detect skin cancer from images of skin lesions. We utilize a diverse dataset of labeled images to train the CNN, enabling it to differentiate between benign and malignant lesions. Additionally, we implement advanced techniques like data augmentation and transfer learning with pre-trained models to enhance accuracy. This model can be integrated into mobile applications, allowing users to upload images of suspicious lesions for preliminary analysis. By providing quick insights, we aim to encourage early medical evaluation and improve overall outcomes in skin cancer detection and treatment.

2. SUPPORTING LITERATURE

2.1. Literature Review

- **Paper 1: Malo, Dipu Chandra, et al. "Skin cancer detection using convolutional neural network."** ***2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*****. IEEE, 2022.**

  In this paper, we have tried to evaluate the chance of deep learning algorithm namely Convolutional Neural Network (CNN) to detect skin cancer classifying benign and malignant mole. We have discussed recent studies that use different models of deep learning on practical datasets to develop the classification process. A detailed workflow to build and run the system is presented too. We have used -Keras and TensorFlow to structure our model. Our proposed VGG-16 model shows a promising development upon some modification to the parameters and classification functions. The model achieves an accuracy of 87.6%. As a result, the study shows a significant outcome of using CNN model in detecting skin cancer.

- **Paper 2: Sharma, Deepti, and Swati Srivastava. "Automatically detection of skin cancer by classification of neural network."** ***International Journal of Engineering and Technical Research*** **4.1 (2016): 15-18.**

The different components in an automated diagnosis of skin cancer include: an automatically skin cancer classification system is developed and the relationship of skin cancer image across different type of neural network are studied with different types of preprocessing. The collected images are feed into the system, and across different image processing procedure to enhance the image properties. Statistical region merging (SRM) algorithm is based on region growing and merging. Then the normal skin is removed from the skin affected area and the cancer cell is left in the image. Useful information can be extracted from these images and pass to the classification system for training and testing. Two neural networks are used as classifier, Back-propagation neural network (BNN) and Auto-associative neural network (AANN). The analysis of work based on MATLAB.

2.1.1. Summary Table

Feature	Paper 1: Malo et al. (2022)	Paper 2: Sharma & Srivastava (2016)
Title	Skin cancer detection using convolutional neural network	Automatically detection of skin cancer by classification of neural network
Publication Venue	2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)	International Journal of Engineering and Technical Research
Objective	Evaluate CNN for detecting and classifying skin cancer (benign vs. malignant)	Develop an automated skin cancer classification system using neural networks
Model Used	VGG-16 with modifications	Back-propagation neural network (BNN) and Auto-associative neural network (AANN)
Frameworks	Keras and TensorFlow	MATLAB
Image Preprocessing Techniques	Not specified in detail	Image enhancement using statistical region merging (SRM) algorithm
Training and Testing Process	Discusses recent studies, detailed workflow provided	Images processed to enhance properties before classification
Results	Achieved accuracy of 87.6%	Analysis based on classification results; specifics not provided
Conclusion	Significant outcome of using CNN in detecting skin cancer	Useful information extraction from processed images for classification

2.2. Findings and Proposals

The papers highlight the significance of automated systems for skin cancer detection, underlining the limitations of current methods, which may lack accuracy or be prohibitively expensive. The proposal is to develop a cost-effective solution utilizing deep learning techniques, specifically convolutional neural networks (CNNs), to improve skin cancer classification.

The first paper emphasizes the effectiveness of various deep learning architectures, with the VGG-16 model achieving an accuracy of 87.6%, showcasing the potential of CNNs in this domain. The second paper discusses the implementation of multiple neural networks, including Back-propagation Neural Network (BNN) and Auto-associative Neural Network (AANN), emphasizing the role of preprocessing techniques like Statistical Region Merging (SRM) to enhance image quality for better classification results.

Our proposed skin cancer detection system will leverage the strengths of convolutional neural networks (CNNs) and transfer learning, focusing on building a robust model trained on a comprehensive dataset of skin lesions. By fine-tuning pre-trained models, we aim to enhance classification accuracy while significantly reducing training time. This innovative system will allow users to upload images for rapid analysis, promoting early detection and improving patient outcomes in skin cancer diagnosis.

3. SYSTEM ANALYSIS

3.1. Analysis of Dataset

3.1.1. About the Dataset

The model will be trained to distinguish between benign and malignant skin lesions using a comprehensive dataset sourced from Kaggle. This dataset consists of labeled images for both classes, with 1800 images of benign lesions and 1497 images of malignant lesions. After performing essential preprocessing, the dataset will be organized into two categories: 'Benign,' containing images of non-cancerous lesions, and 'Malignant,' including images of cancerous lesions.

3.1.2. Explore the Dataset

Dataset consist of 3297 images labeled across 2 classes – benign and Malignant. 1800 belong to the label of benign roads and 1497 to the other.

The datasets are split into 70:30 for training and testing respectively duringmodel training.

Loading the dataset.

```
datadir = r"C:\Users\HP\Desktop\Skin Cancer Detection\archivemb\train"

def get_training_data(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_path = os.path.join(path, img)
                img_arr = cv2.imread(img_path, cv2.IMREAD_COLOR)  # Read RGB images
                if img_arr is None:
                    print(f"Error reading image: {img_path}")
                    continue
                resized_arr = cv2.resize(img_arr, (img_size, img_size))  # Resize the image
                resized_arr = cv2.cvtColor(resized_arr, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
                data.append([resized_arr.astype('float32'), class_num])  # Convert to float32 for consistency
            except Exception as e:
                print(f"Error processing image {img_path}: {e}")
    return np.array(data, dtype=object)  # Use dtype=object if data types might be mixed

# Example usage
training_data = get_training_data(datadir)
```

Class labels

```
labels = ['benign', 'malignant'] # Define your class labels
img_size = 64
```

Splitting Training and Testing Data

```
from sklearn.model_selection import train_test_split

X = []
y = []

for feature, label in train:
    X.append(feature)
    y.append(label)

for feature, label in test:
    X.append(feature)
    y.append(label)


# resize data for deep learning
X = np.array(X).reshape(-1, img_size, img_size, 3)
y = np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=32)
```

3.2. Data Pre Processing

The images in the skin cancer detection dataset vary in size and color profiles. To provide them as input to a convolutional neural network (CNN) architecture, it is essential to convert them into a fixed size, specifically 64 x 64 x 3. The OpenCV Python library is utilized for preprocessing tasks such as resizing and color conversion. Images in the dataset are resized to this standard size using the OpenCV resize function, ensuring uniformity across the dataset for effective training and evaluation of the model

3.2.1. Resizing the Image – 64 x 64 x 3

```
def get_training_data(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_path = os.path.join(path, img)
                img_arr = cv2.imread(img_path, cv2.IMREAD_COLOR)  # Read RGB images
                if img_arr is None:
                    print(f"Error reading image: {img_path}")
                    continue
                resized_arr = cv2.resize(img_arr, (img_size, img_size))  # Resize the image
                resized_arr = cv2.cvtColor(resized_arr, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
                data.append([resized_arr.astype('float32'), class_num])  # Convert to float32 for consistency
            except Exception as e:
                print(f"Error processing image {img_path}: {e}")
    return np.array(data, dtype=object)  # Use dtype=object if data types might be mixed
```

3.2.2. Analysis of Feature Variable

The feature list consists of attributes that describe the images in the dataset. All images are in "jpg" format and are resized to a uniform size of 64 x 64 pixels for both training and testing.

Dimensions: 64 x 64 x 3

Bit depth: 24

3.2.3. Analysis of Class Variables

Classes are sometimes called as targets/labels or categories. Class of the dataset is the category to which the input will be classified to. That means the finalresult of an application. Here we have to class variables: Benign and Malignant. The images are classified into these 2 classes.

```
labels = ['benign', 'malignant'] # Define your class labels
img_size = 64
```

3.3. Data Visualization

The images are resized to the size 64 x 64 x 3 and images displayed alongwith their labels or class variables.

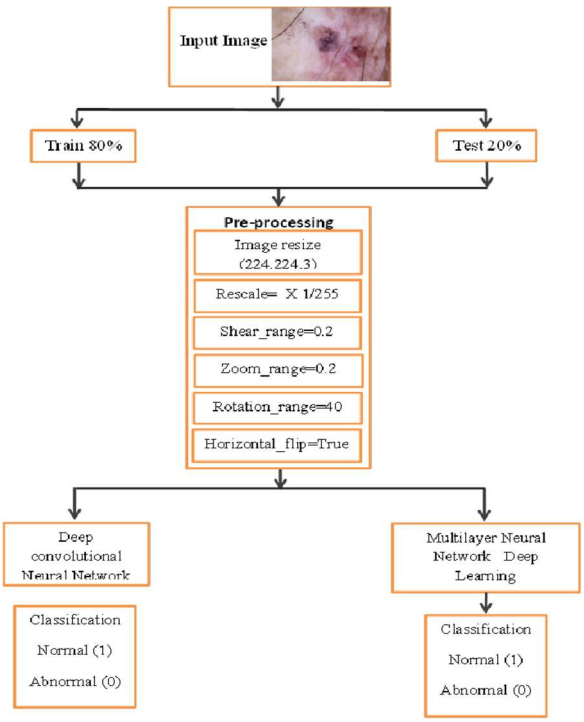
3.4. Analysis of Architecture

This project aims to develop a machine learning model for the detection of skin cancer using images of skin lesions. Given the critical nature of early diagnosis in skin cancer treatment, our approach leverages advanced deep learning techniques, specifically Convolutional Neural Networks (CNNs) and transfer learning with pre-trained models. The use of transfer learning allows us to harness the power of models trained on large datasets, thereby improving accuracy and reducing the need for extensive training data.

Model 1 :Convolutional Neural Network(CNN)

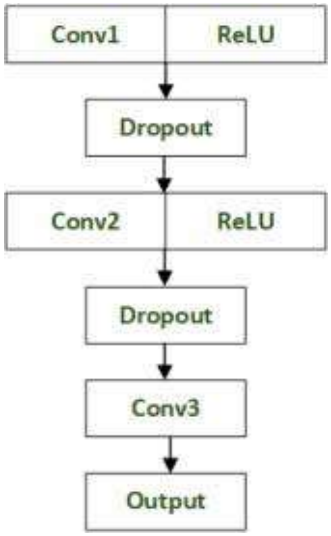
The core of our project is a custom CNN model specifically designed for binary classification of skin lesions into "benign" and "malignant" categories. The architecture consists of several convolutional layers followed by pooling layers, which extract and downsample features from the input images.

3.4.1. Block Diagram



This architecture is designed with 48 convolution layers, along with Max Pooling, Average Pooling, Dropout, and Fully Connected layers, providing robust feature

extraction and efficient classification of the images.



Because of the framework that CNN presented it was made possible to train ultra- deep neural networks and by that i mean that i network can contain hundreds or thousands of layers and still achieve great performance. The CNN were initially applied to the image recognition task but as it is mentioned in the paper that the framework can also be used for non-computer vision tasks also to achieve better accuracy. Many of you may argue that simply stacking more layers also gives usbetter accuracy why was there a need of Residual learning for training ultra-deepneural networks.

- A convoultion with a kernel size of 7 * 7 and 64 different kernels all with a stride of size 2 giving us 1 layer.
- Next we see max pooling with also a stride size of 2.
- In the next convolution there is a 1 * 1,64 kernel following this a 3 * 3,64 kernel and at last a 1 * 1,256 kernel, These three layers are repeated in total 3 time so giving us 9 layers in this step.
- Next we see kernel of 1 * 1,128 after that a kernel of 3 * 3,128 and at last a kernel of 1 * 1,512 this step was repeated 4 time so giving us 12 layers in this step.
- After that there is a kernel of 1 * 1,256 and two more kernels with 3 * 3,256 and 1 * 1,1024 and this is repeated 6 time giving us a total of 18 layers.
- And then again a 1 * 1,512 kernel with two more of 3 * 3,512 and 1 * 1,2048 and this

was repeated 3 times giving us a total of 9 layers.

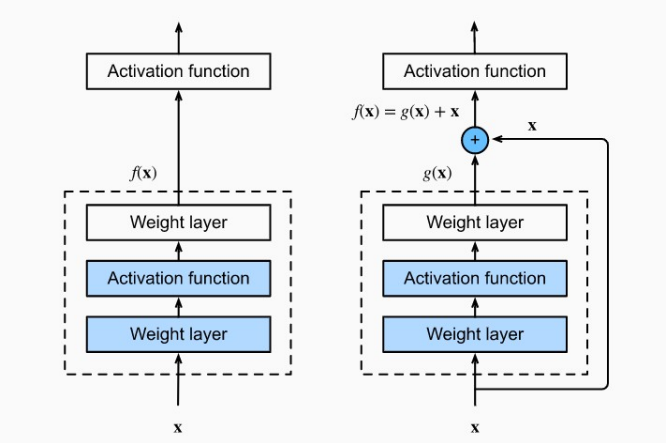
- After that we do a average pool and end it with a fully connected layer containing 1000 nodes and at the end a sigmoid function so this gives us 1 layer.
- We don't actually count the activation functions and the max/ average pooling layers.

So total this it gives us a 1 + 9 + 12 + 18 + 9 + 1 = 50 layers Deep Convolutional network.

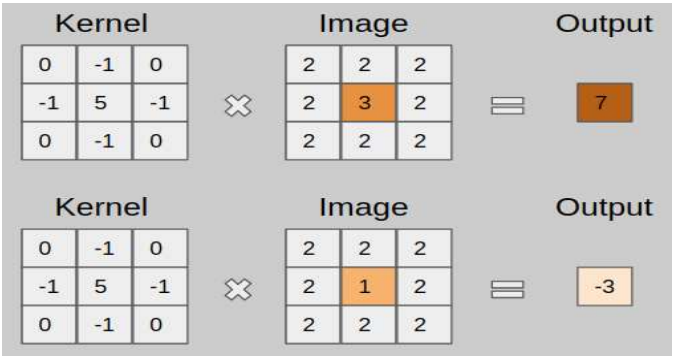
Here in our system, Avg Pool layer if followed by a drop out layer, then 4 dense block layer, which optimises the number of values in the fully connected layer to 2, which is the count of the class variable of our system. So the final output will be an array of length 2.

3.4.2. Diagrams and Details of Each Layer

Convolutional Layer: The key building block in a convolutional neural network is the convolutionallayer. This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to thesize of the filter (MxM). The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image. Size of the feature map = [(input_ size — kernel size + 2 x padding) / stride] +1.



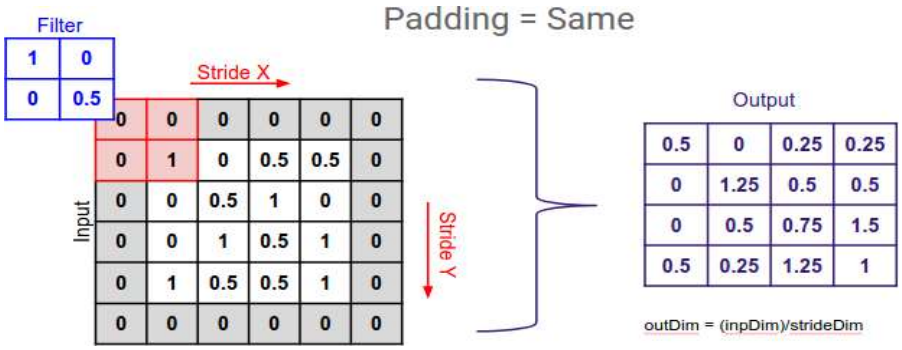
Kernel/Filter: In Convolutional neural network, the kernel is nothing but a filter that is used to extract the features from the images. The kernel is a matrix thatmoves over the input data, performs the dot product with the sub-region of input data, and gets the output as the matrix of dot products. Kernel moves on the input data by the stride value. If the stride value is 2, then kernel moves by 2 columns of pixels in the input matrix. In short, the kernel is used to extract high-level features like edges from the image.



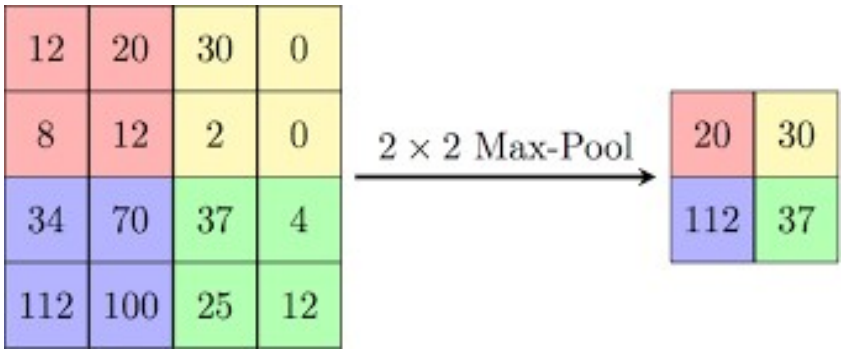
Stride: Stride is a component of convolutional neural networks, or neuralnetworks tuned for the compression of images and video data. Stride is a parameter of the neural network‘s filter that modifies the amount of movement over the image or video. For example, if a neuralnetwork's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than afraction or decimal

1	0	1	0	-1	0	-1
1	0	1	0	-1	0	-1
1	0	1	0	-1	0	-1
255	255	255	0	0	0	0
255	255	255	0	0	0	0
255	255	255	0	0	0	0

Padding: Padding is a term relevant to convolutional neural network as it refers to the numberof pixels added to an image when it is being processed by the kernel of a CNN. For example, if the padding in a CNN is set to zero, then every pixel value that is added will be of value zero. If, however, the zero padding is set to one, there will be a one-pixel border added to theimage with a pixel value of zero. A Finer Padding value is same.



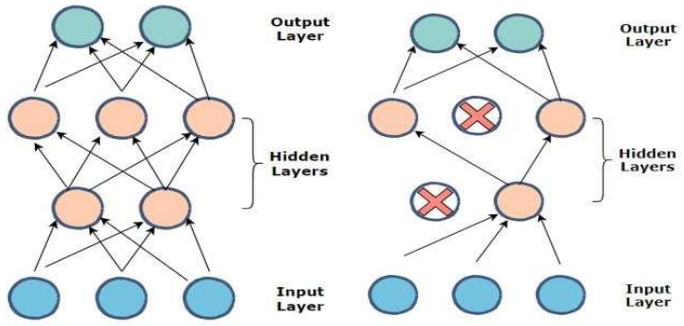
Pooling Layer: The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending



upon method used, there are several types of Pooling operations. In Max Pooling, the

largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section are computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the Fully Connected Layer. The pooling layer is also called subsampling layer. Max pooling provides much better performance than average pooling. In max pooling layer, the maximum value among all the values in a matrix is chosen. Here we are using MaxPooling.

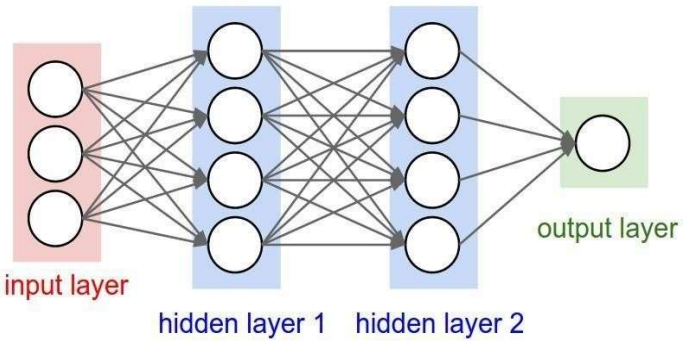
Dropout Layer: Another typical characteristic of CNNs is a Dropout layer. Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on thetraining data causing a negative impact in the model‘s performance when used on a new data. To overcome this problem, a dropout layer is utilized wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropoutof 0.5, 50% of the nodes are dropped out randomly from the neural network.



Original network drop out networks

Fully connected Layer: They typically were included as the last few layers of most CNNs, appearing after several convolution and subsampling operations were performed. Fully connected layers were independent neural networks that possessed one or more hidden layers. Their operations involved multiplying their inputs by trainable weight vectors, with a trainable bias sometimes summed to those results. The output of these layers was traditionally sent through activation functions, similarly to

convolution layers. These layers take the output of the previous layers, flattens them and turns them into a single vector that can be an input for the next stage.



Activation Functions: Finally, one of the most important parameters of the CNN model is theactivation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network. It adds non-linearityto the network. There are several commonly used activation functions such as the ReLU, SoftMax, tanH and the Sigmoid functions.For a binary classification CNN model, sigmoid and SoftMax functions are preferred and for multi-class classification, generally Sigmoid us used.In our work we used two activation functions ReLu, and Sigmoid.

ReLU (rectified linear activation function) is a linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance. The usage of ReLU helps to prevent the exponential growth in the computation required to operate the neural network.

Sigmoid function is a mathematical function commonly used in machine learning, particularly in binary classification tasks. It maps any real-valued number to a value between 0 and 1, making it useful for modeling probabilities. The function is defined as σ(x)=11+e−x\sigma(x) = \frac{1}{1 + e^{-x}}σ(x)=1+e−x1, where eee is the base of the natural logarithm. In the context of neural networks, the sigmoid

function is often used as an activation function in the output layer for binary classification problems. By outputting a single value, the sigmoid function allows the model to represent the probability that a given input belongs to the positive class, with a threshold (commonly set at 0.5) used to determine class membership. Despite its popularity, the sigmoid function can lead to issues such as vanishing gradients, especially in deep networks, prompting the use of alternative activation functions like ReLU in hidden layers.

Model 2: Transfer Learning with ResNet50

ResNet50 is a highly popular pre-trained CNN architecture, known for introducing residual connections, which address the problem of vanishing gradients in deep networks. The network is 50 layers deep and is pre-trained on the ImageNet dataset, giving it a powerful ability to generalize features across many image types.

1. Pre-trained Base (ResNet50):
   - The ResNet50 model is imported without the top layer (include_top=False), keeping the layers pre-trained on ImageNet.
   - Frozen Layers: All layers of the base ResNet50 model are initially frozen to retain the learned features from ImageNet.
2. GlobalAveragePooling2D:
   - Purpose: Converts the feature maps from the pre-trained network to a 1D vector by averaging the values of each feature map. This reduces the dimensionality without losing important spatial information.
3. Fully Connected Layer 1:
   - Dense Layer: 128 neurons.
   - Activation: ReLU.
   - Dropout: 0.2 to reduce overfitting.
4. Output Layer:

- Dense Layer: 1 neuron.
- Activation: Sigmoid for binary classification.

5. Fine-tuning:

- After initial training with frozen layers, fine-tuning is performed by unfreezing some of the deeper layers of ResNet50 (closer to the output) to allow for more task-specific learning.

6. Advantages:

- Leverages pre-trained features learned from the vast ImageNet dataset, leading to faster convergence and higher accuracy.
- Residual connections in ResNet50 help in training very deep networks effectively by addressing vanishing gradients.
- Ideal for this skin cancer dataset, where deep features from a general pre-trained network can be applied to specific tasks like skin lesion classification.

7. Training Results:

- Accuracy: 81.97%
- Generalization: The model shows a good ability to generalize well to unseen data, thanks to pre-trained features.

Model 3: Transfer Learning with VGG16

VGG16 is another popular CNN architecture, consisting of 16 layers. It is simpler than ResNet50 in structure but known for its uniform architecture of stacked convolutional layers. VGG16 is pre-trained on ImageNet, which makes it a valuable feature extractor for image classification tasks.

1. Pre-trained Base (VGG16):

- The VGG16 model is loaded without the top layers (include_top=False) and with ImageNet weights.
- Frozen Layers: Initially, all the layers of VGG16 are frozen to retain the features

learned from ImageNet.

2. Flatten Layer:

- Purpose: Unlike ResNet50, VGG16 outputs 2D feature maps. The Flatten layer converts these 2D maps into a 1D vector to prepare for fully connected layers.

3. Fully Connected Layer 1:

- Dense Layer: 128 neurons.
- Activation: ReLU.
- Dropout: 0.3 to prevent overfitting.

4. Output Layer:

- Dense Layer: 1 neuron.
- Activation: Sigmoid for binary classification.

5. Fine-tuning:

- Fine-tuning is performed similarly to ResNet50 by unfreezing some deeper layers after training the top layers. This helps adjust the model more specifically to the skin cancer dataset while retaining the power of pre-trained features.

6. Advantages:

- Simpler architecture compared to ResNet50 but still highly effective for feature extraction.
- Pre-trained weights on ImageNet make it robust for small datasets, especially for tasks like skin lesion detection.

7. Training Results:

- Accuracy: 84.24%
- Performance: Slightly less accurate than ResNet50, but still highly competitive, especially for tasks where fewer layers are preferable due to computation constraints.

3.4.3. Dimension Table

Model Name	Dimension Name	Description
Custom CNN	Input Size	Size of the input images (width x height)
Custom CNN	Class Labels	Labels for classification (e.g., benign, malignant)
Custom CNN	Number of Layers	Total layers in the model
ResNet50	Input Shape	Shape of the input tensor for the model
ResNet50	Class Labels	Labels for classification (e.g., benign, malignant)
ResNet50	Pretrained Weights	Weights used for initialization
VGG16	Input Shape	Shape of the input tensor for the model
VGG16	Class Labels	Labels for classification (e.g., benign, malignant)
VGG16	Pretrained Weights	Weights used for initialization

3.4.4 Project Pipeline

3.4.5 Feasibility Analysis

A feasibility study aims to objectively and rationally uncover the strengths and weaknesses of an existing system or proposed system, opportunities and threats present in the natural environment, the resources required to carry through, and ultimately the prospects for success.

Evaluated the feasibility of the system in terms of the following categories:

- Technical Feasibility
- Economical Feasibility

- Operational Feasibility

Technical Feasibility

The application for **Skin Cancer Detection** is technically feasible because all the necessary resources for development and operation are readily available and reliable. The hardware requirements include a computer system with internet access. The code is implemented in Jupyter Notebook, which allows for the creation of a dedicated environment with all required libraries installed. These requirements are easily accessible and dependable, resulting in a system that saves time and minimizes the need for extensive manpower. The application will be easy to develop, manage, and modify, as the technologies employed are common and widely available.

Economic Feasibility

The cost to manage this system will be lesser. The system requires only a computer for working. The code is working on Jupyter Notebook, so it consumes no amount of internet. It will be economically feasible. And the money spend for the application will be worth.

Operational Feasibility

The developed system is completely driven and user friendly. Since the code is written on Jupyter Notebook, the system has a separate environment without having to utilize resources in a common environment. There is no need of skill for a new user to open this application and use it. The interface contains only a file upload option and a Detect button. Users also need to be aware of the application initially. Then they can use it easily. So, it is feasible.

3.4.6 System Environment

System environment specifies the hardware and software configuration of the new system. Regardless of how the requirement phase proceeds, it ultimately ends with the software requirement specification. A good SRS contains all the system requirements to a level of detail sufficient to enable designers to design a system that satisfies those requirements. The system specified in the SRS will assist the potential users to determine if the system meets their needs or how the system must bemodified to meet their needs

3.4.6.1 Software Environment

Various software used for the development of this application are the following:

Python: Python is a high-level programming language that lets developers workquickly and integrate systems more efficiently. This application is developed by using many of the Python libraries and packages such as:

- **Matplotlib**: is a cross-platform, data visualization and graphical plotting library for Python. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. In this application, its used for plotting the images from datasets as well as to plot the training graph
- **NumPy** is a Python library used for working with arrays. NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objectsand a collection of routines for processing those arrays. In this application, its used for handling arrays and reshaping.
- **TensorFlow** is an open-source library developed by Google primarily for deep learning applications. In this application, its used for creating and handling the model.
- **Keras** is a powerful and easy-to-use free open-source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network

models in just a few lines of code. In this application, its used for creating and handling and saving the model.

- The OS module in Python provides functions for interacting with the operating system. OS comes under Python‘s standard utility modules. This module provides a portable way of using operating system-dependent functionality.
- OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. The library has more than 2500 optimized algorithms. In this application, its used for reading and saving images.
- **Streamlit:**

Streamlit is an open-source framework used for building interactive web applications specifically for data science and machine learning projects. It allows users to create custom user interfaces with minimal code, enabling them to visualize data, display results,and interact with machine learning models easily. Streamlit automatically handles user input and updates the application in real time, making it a powerful tool for quickly prototyping and sharing data-driven applications.

- **Github**

Git is an open-source version control system that was started by Linus Torvalds. Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute. Git is the preferred version control system of most developers, since it has multiple advantages over the other systems available. It stores file changes more efficiently and ensures file integrity better.

The social networking aspect of GitHub is probably its most powerful feature, allowing projects to grow more than just about any of the other features offered. Project revisions can be discussed publicly, so a mass of experts can contribute knowledge and collaborate to advance a project forward.

3.4.6.2 Hardware Environment

Selection of hardware configuration is very important task related to the software development

3.4.6.2.1 Processor: 2 GHz or faster (dual-core or quad-core will be much faster)

3.4.6.2.2 Memory: 8 GB RAM or greater

3.4.6.2.3 Disk space: 40 GB or greater good internet connectivity

3.4.6.2.4 GPU :2GB

4. SYSTEM DESIGN

4.1. Model Building

4.1.1 Model Planning

The Skin Cancer detection dataset consist of 2 folders which contains Benign images and Malignant images. Both the labels are divided into training and testing datasets in a ratio of 70:30. The model is trained using the training dataset and tested against the test data.

```
model = Sequential()

model.add(Conv2D(64, (5, 5), input_shape=(img_size, img_size, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(128, (5, 5), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(256, (5, 5), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(16, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
```

```
# loading the model
pretrained_model = tf.keras.applications.ResNet50(include_top=False, weights='imagenet', input_shape=(img_size,img_size , 3))
```

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_size, img_size, 3))
```

4.1.2 . Training

In this project, three models are used: a custom CNN model, ResNet50, and VGG16 as the base pretrained models. After each base model, a dropout layer and 4 dense layers are added to create the desired architecture. The final fully connected layer has a single output with a sigmoid activation function, suitable for binary classification.

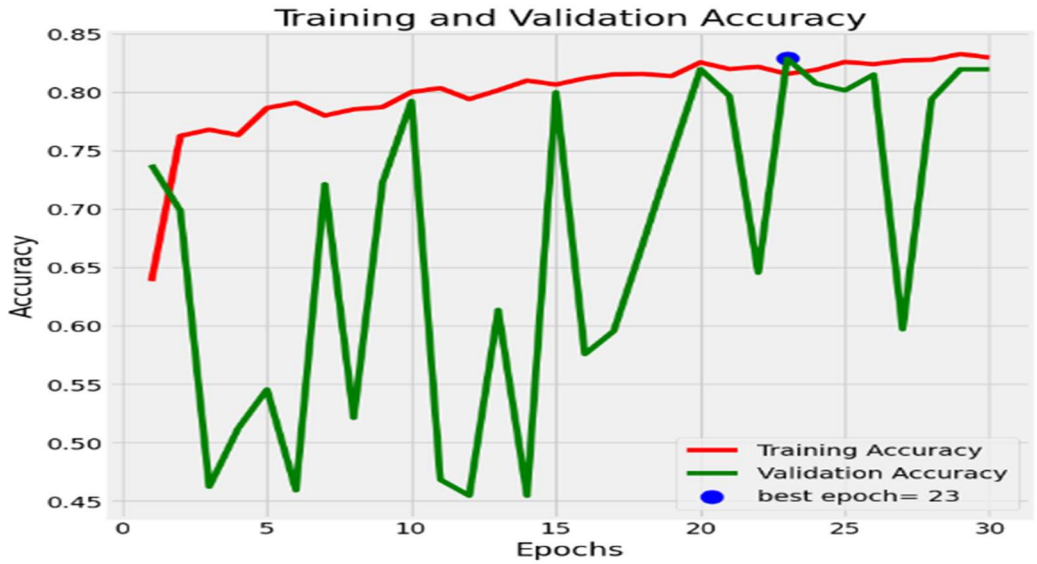
Custom CNN Model:-

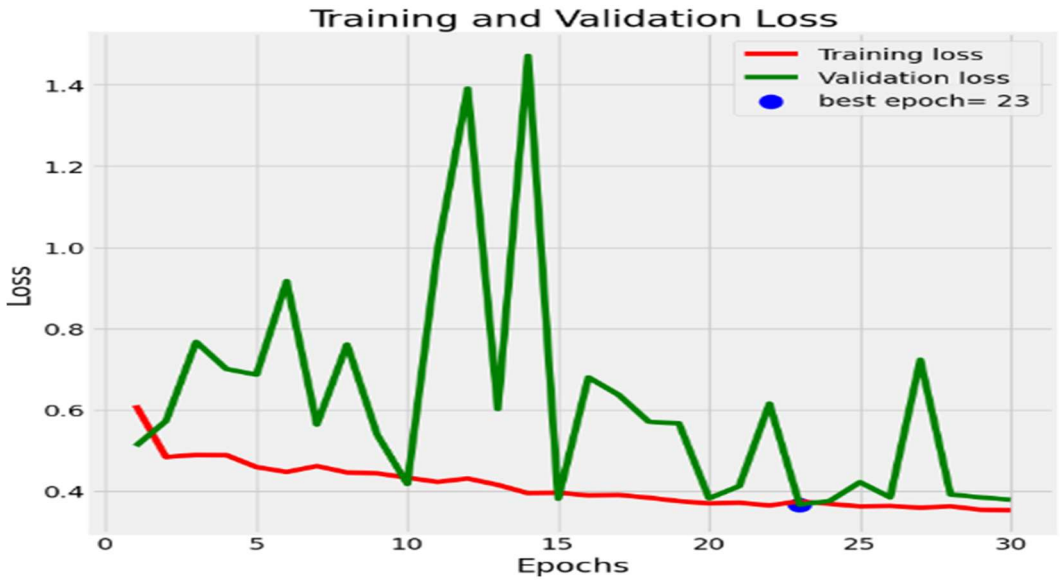
```
history = model.fit(datagen.flow(X_train, y_train, batch_size=32), callbacks=[lr_reduction], validation_data=(X_test, y_test), epochs=30)

# Evaluate the model
scores = model.evaluate(X_test, y_test, verbose=0)
print(f"Accuracy: {scores[1]*100:.2f}% \nError: {(1-scores[1])*100:.2f}%")
```

Training is performed based on different parameters like, epoch, batch size, learning rate etc.

- The number of epoch is set to 30 initially with a batch size of 32.
- Learning Rate : 0.00001
- The model gains an accuracy of 81.97% and a loss of 18.03%.

Training data is the initial dataset we use to teach an application to recognize pattern. Training dataset is used to train our model, in order to get correct predictions by the model. Images are classified into two different folders, where folder name denotes the class. By running the 30 epochs to generate the model, and achieved an accuracy of 81.97% and a loss of 18.03%.





Filters/neurons detect spatial patterns such as edges in an image by detecting the changes in intensity values of the image. Filter size is then size of matrix using in the CNN layers. Stride is the number of pixels shifts over the input matrix. Padding is used sometimeswhen filter does not perfectly the input image, then we add zeros to make it fit. We set padding to SAME so that the input image gets fully covered by the filter and specified stride. It is called SAME because, for stride 1, the output will be the same as the input. The featuremap captures the results of applying the filters to aninput image. I.e., at each layer, the featuremap is the output of that layer.

```
Epoch 1/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 69s 700ms/step - accuracy: 0.5642 - loss: 0.6747 - val_accuracy: 0.7379 - val_loss: 0.5104 - learning_rate: 0.0010
Epoch 2/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 56s 672ms/step - accuracy: 0.7582 - loss: 0.4822 - val_accuracy: 0.6985 - val_loss: 0.5720 - learning_rate: 0.0010
Epoch 3/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 55s 658ms/step - accuracy: 0.7697 - loss: 0.4879 - val_accuracy: 0.4621 - val_loss: 0.7670 - learning_rate: 0.0010
Epoch 4/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 619ms/step - accuracy: 0.7662 - loss: 0.4807
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
83/83 ━━━━━━━━━━━━━━━━━━━━ 56s 668ms/step - accuracy: 0.7661 - loss: 0.4808 - val_accuracy: 0.5121 - val_loss: 0.7002 - learning_rate: 0.0010
Epoch 5/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 55s 665ms/step - accuracy: 0.7760 - loss: 0.4679 - val_accuracy: 0.5455 - val_loss: 0.6861 - learning_rate: 5.0000e-04
Epoch 6/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 55s 658ms/step - accuracy: 0.7978 - loss: 0.4312 - val_accuracy: 0.4591 - val_loss: 0.9163 - learning_rate: 5.0000e-04
Epoch 7/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 605ms/step - accuracy: 0.7841 - loss: 0.4459
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 654ms/step - accuracy: 0.7841 - loss: 0.4460 - val_accuracy: 0.7212 - val_loss: 0.5632 - learning_rate: 5.0000e-04
Epoch 8/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 655ms/step - accuracy: 0.7839 - loss: 0.4468 - val_accuracy: 0.5212 - val_loss: 0.7607 - learning_rate: 2.5000e-04
Epoch 9/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 56s 679ms/step - accuracy: 0.7819 - loss: 0.4541 - val_accuracy: 0.7227 - val_loss: 0.5383 - learning_rate: 2.5000e-04
Epoch 10/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 653ms/step - accuracy: 0.7842 - loss: 0.4532 - val_accuracy: 0.7924 - val_loss: 0.4168 - learning_rate: 2.5000e-04
Epoch 11/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 55s 664ms/step - accuracy: 0.7847 - loss: 0.4627 - val_accuracy: 0.4682 - val_loss: 0.9923 - learning_rate: 2.5000e-04
Epoch 12/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 55s 656ms/step - accuracy: 0.7973 - loss: 0.4393 - val_accuracy: 0.4545 - val_loss: 1.3907 - learning_rate: 2.5000e-04
Epoch 13/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 612ms/step - accuracy: 0.7975 - loss: 0.4085
Epoch 13: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
83/83 ━━━━━━━━━━━━━━━━━━━━ 55s 658ms/step - accuracy: 0.7975 - loss: 0.4085 - val_accuracy: 0.6136 - val_loss: 0.6014 - learning_rate: 2.5000e-04
Epoch 14/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 55s 660ms/step - accuracy: 0.8120 - loss: 0.3967 - val_accuracy: 0.4545 - val_loss: 1.4715 - learning_rate: 1.2500e-04
Epoch 15/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 655ms/step - accuracy: 0.8058 - loss: 0.4029 - val_accuracy: 0.8000 - val_loss: 0.3805 - learning_rate: 1.2500e-04
```

First 15 epochs of model training

```
Epoch 16/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 56s 670ms/step - accuracy: 0.8114 - loss: 0.3867 - val_accuracy: 0.5758 - val_loss: 0.6795 - learning_rate: 1.2500e-04
Epoch 17/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 644ms/step - accuracy: 0.8224 - loss: 0.3768 - val_accuracy: 0.5955 - val_loss: 0.6367 - learning_rate: 1.2500e-04
Epoch 18/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 596ms/step - accuracy: 0.8134 - loss: 0.3800
Epoch 18: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
83/83 ━━━━━━━━━━━━━━━━━━━━ 53s 643ms/step - accuracy: 0.8135 - loss: 0.3800 - val_accuracy: 0.6697 - val_loss: 0.5703 - learning_rate: 1.2500e-04
Epoch 19/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 55s 657ms/step - accuracy: 0.8100 - loss: 0.3783 - val_accuracy: 0.7455 - val_loss: 0.5662 - learning_rate: 6.2500e-05
Epoch 20/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 646ms/step - accuracy: 0.8335 - loss: 0.3605 - val_accuracy: 0.8197 - val_loss: 0.3817 - learning_rate: 6.2500e-05
Epoch 21/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 608ms/step - accuracy: 0.8163 - loss: 0.3735
Epoch 21: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
83/83 ━━━━━━━━━━━━━━━━━━━━ 55s 657ms/step - accuracy: 0.8164 - loss: 0.3735 - val_accuracy: 0.7970 - val_loss: 0.4118 - learning_rate: 6.2500e-05
Epoch 22/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 648ms/step - accuracy: 0.8284 - loss: 0.3722 - val_accuracy: 0.6455 - val_loss: 0.6150 - learning_rate: 3.1250e-05
Epoch 23/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 654ms/step - accuracy: 0.8152 - loss: 0.3733 - val_accuracy: 0.8288 - val_loss: 0.3663 - learning_rate: 3.1250e-05
Epoch 24/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 56s 668ms/step - accuracy: 0.8148 - loss: 0.3672 - val_accuracy: 0.8076 - val_loss: 0.3753 - learning_rate: 3.1250e-05
Epoch 25/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 654ms/step - accuracy: 0.8134 - loss: 0.3718 - val_accuracy: 0.8015 - val_loss: 0.4214 - learning_rate: 3.1250e-05
Epoch 26/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 601ms/step - accuracy: 0.8285 - loss: 0.3586
Epoch 26: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 649ms/step - accuracy: 0.8285 - loss: 0.3586 - val_accuracy: 0.8152 - val_loss: 0.3841 - learning_rate: 3.1250e-05
Epoch 27/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 53s 635ms/step - accuracy: 0.8331 - loss: 0.3554 - val_accuracy: 0.5970 - val_loss: 0.7231 - learning_rate: 1.5625e-05
Epoch 28/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 53s 636ms/step - accuracy: 0.8364 - loss: 0.3654 - val_accuracy: 0.7939 - val_loss: 0.3913 - learning_rate: 1.5625e-05
Epoch 29/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 602ms/step - accuracy: 0.8322 - loss: 0.3386
Epoch 29: ReduceLROnPlateau reducing learning rate to 1e-05.
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 648ms/step - accuracy: 0.8322 - loss: 0.3388 - val_accuracy: 0.8197 - val_loss: 0.3840 - learning_rate: 1.5625e-05
Epoch 30/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 54s 645ms/step - accuracy: 0.8216 - loss: 0.3510 - val_accuracy: 0.8197 - val_loss: 0.3783 - learning_rate: 1.0000e-05
Accuracy: 81.97%
Error: 18.03%
```

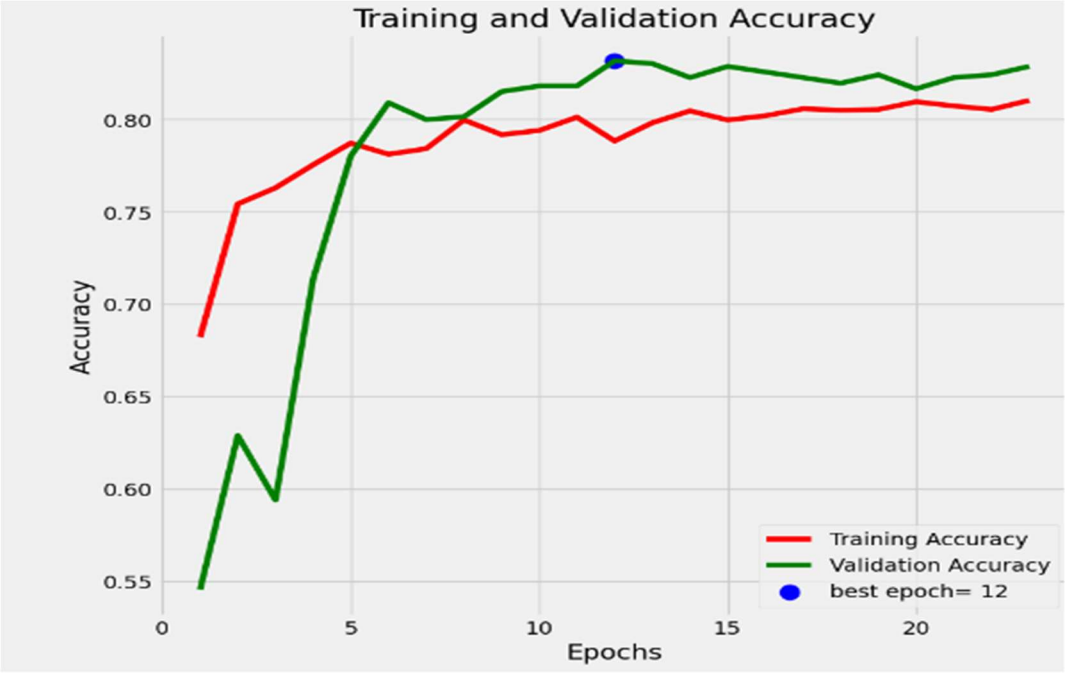
16th to 30th epochs of model training

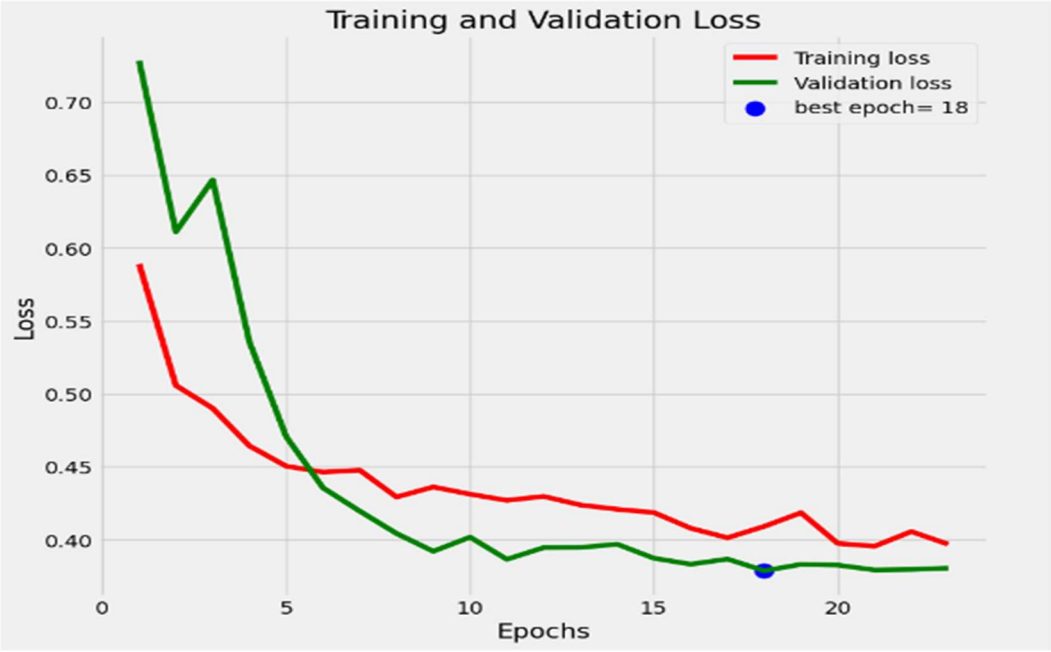
ResNet50 Model:-

```
# Train the model with both callbacks
history = model.fit(datagen.flow(X_train, y_train, batch_size=32),
                    validation_data=(X_test, y_test),
                    callbacks=[early_stopping, model_checkpoint, lr_reduction],
                    epochs=25)
```

Training is performed based on different parameters like, epoch, batch size, learning rate etc.

- The number of epoch is set to 25 initially with a batch size of 32.
- Learning Rate : 0.00001
- The model gains an accuracy of 81.97% and a loss of 18.03%.





```
Epoch 1/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 290ms/step - accuracy: 0.6208 - loss: 0.6352
Epoch 1: val_loss improved from inf to 0.72840, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 56s 471ms/step - accuracy: 0.6215 - loss: 0.6346 - val_accuracy: 0.5455 - val_loss: 0.7284 - learning_rate: 0.0010
Epoch 2/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 284ms/step - accuracy: 0.7583 - loss: 0.5062
Epoch 2: val_loss improved from 0.72840 to 0.61115, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 32s 380ms/step - accuracy: 0.7582 - loss: 0.5062 - val_accuracy: 0.6288 - val_loss: 0.6111 - learning_rate: 0.0010
Epoch 3/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 286ms/step - accuracy: 0.7704 - loss: 0.4875
Epoch 3: val_loss did not improve from 0.61115
83/83 ━━━━━━━━━━━━━━━━━━━━ 30s 357ms/step - accuracy: 0.7703 - loss: 0.4876 - val_accuracy: 0.5939 - val_loss: 0.6467 - learning_rate: 0.0010
Epoch 4/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 284ms/step - accuracy: 0.7719 - loss: 0.4661
Epoch 4: val_loss improved from 0.61115 to 0.53559, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 31s 378ms/step - accuracy: 0.7719 - loss: 0.4661 - val_accuracy: 0.7136 - val_loss: 0.5356 - learning_rate: 0.0010
Epoch 5/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 294ms/step - accuracy: 0.7935 - loss: 0.4470
Epoch 5: val_loss improved from 0.53559 to 0.47065, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 33s 401ms/step - accuracy: 0.7934 - loss: 0.4470 - val_accuracy: 0.7803 - val_loss: 0.4707 - learning_rate: 0.0010
Epoch 6/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 289ms/step - accuracy: 0.7812 - loss: 0.4480
Epoch 6: val_loss improved from 0.47065 to 0.43556, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 32s 384ms/step - accuracy: 0.7812 - loss: 0.4480 - val_accuracy: 0.8091 - val_loss: 0.4356 - learning_rate: 0.0010
Epoch 7/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 279ms/step - accuracy: 0.7834 - loss: 0.4496
Epoch 7: val_loss improved from 0.43556 to 0.41957, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 31s 374ms/step - accuracy: 0.7834 - loss: 0.4496 - val_accuracy: 0.8000 - val_loss: 0.4196 - learning_rate: 0.0010
Epoch 8/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 277ms/step - accuracy: 0.8041 - loss: 0.4231
Epoch 8: val_loss improved from 0.41957 to 0.40424, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 30s 366ms/step - accuracy: 0.8041 - loss: 0.4232 - val_accuracy: 0.8015 - val_loss: 0.4042 - learning_rate: 0.0010
Epoch 9/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 287ms/step - accuracy: 0.7988 - loss: 0.4411
Epoch 9: val_loss improved from 0.40424 to 0.39202, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 32s 386ms/step - accuracy: 0.7988 - loss: 0.4411 - val_accuracy: 0.8152 - val_loss: 0.3920 - learning_rate: 0.0010
Epoch 10/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step - accuracy: 0.8064 - loss: 0.4105
Epoch 10: val_loss did not improve from 0.39202
83/83 ━━━━━━━━━━━━━━━━━━━━ 29s 350ms/step - accuracy: 0.8062 - loss: 0.4107 - val_accuracy: 0.8182 - val_loss: 0.4017 - learning_rate: 0.0010
```

First 10 epochs of model training

```
Epoch 11/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 278ms/step - accuracy: 0.7883 - loss: 0.4330
Epoch 11: val_loss improved from 0.39202 to 0.38652, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 31s 372ms/step - accuracy: 0.7884 - loss: 0.4330 - val_accuracy: 0.8182 - val_loss: 0.3865 - learning_rate: 0.0010
Epoch 12/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step - accuracy: 0.7733 - loss: 0.4444
Epoch 12: val_loss did not improve from 0.38652
83/83 ━━━━━━━━━━━━━━━━━━━━ 29s 351ms/step - accuracy: 0.7734 - loss: 0.4442 - val_accuracy: 0.8318 - val_loss: 0.3945 - learning_rate: 0.0010
Epoch 13/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 299ms/step - accuracy: 0.8134 - loss: 0.4086
Epoch 13: val_loss did not improve from 0.38652
83/83 ━━━━━━━━━━━━━━━━━━━━ 31s 370ms/step - accuracy: 0.8132 - loss: 0.4088 - val_accuracy: 0.8303 - val_loss: 0.3946 - learning_rate: 0.0010
Epoch 14/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 310ms/step - accuracy: 0.7882 - loss: 0.4321
Epoch 14: val_loss did not improve from 0.38652

Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
83/83 ━━━━━━━━━━━━━━━━━━━━ 32s 381ms/step - accuracy: 0.7884 - loss: 0.4320 - val_accuracy: 0.8227 - val_loss: 0.3969 - learning_rate: 0.0010
Epoch 15/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 291ms/step - accuracy: 0.7910 - loss: 0.4314
Epoch 15: val_loss did not improve from 0.38652
83/83 ━━━━━━━━━━━━━━━━━━━━ 30s 361ms/step - accuracy: 0.7911 - loss: 0.4312 - val_accuracy: 0.8288 - val_loss: 0.3873 - learning_rate: 5.0000e-04
Epoch 16/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 287ms/step - accuracy: 0.7837 - loss: 0.4330
Epoch 16: val_loss improved from 0.38652 to 0.38316, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 32s 385ms/step - accuracy: 0.7840 - loss: 0.4327 - val_accuracy: 0.8258 - val_loss: 0.3832 - learning_rate: 5.0000e-04
Epoch 17/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 294ms/step - accuracy: 0.8135 - loss: 0.3913
Epoch 17: val_loss did not improve from 0.38316
83/83 ━━━━━━━━━━━━━━━━━━━━ 30s 364ms/step - accuracy: 0.8134 - loss: 0.3914 - val_accuracy: 0.8227 - val_loss: 0.3867 - learning_rate: 5.0000e-04
Epoch 18/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 280ms/step - accuracy: 0.7999 - loss: 0.4276
Epoch 18: val_loss improved from 0.38316 to 0.37871, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 31s 375ms/step - accuracy: 0.7999 - loss: 0.4274 - val_accuracy: 0.8197 - val_loss: 0.3787 - learning_rate: 5.0000e-04
Epoch 19/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 281ms/step - accuracy: 0.8201 - loss: 0.3899
Epoch 19: val_loss did not improve from 0.37871
83/83 ━━━━━━━━━━━━━━━━━━━━ 29s 350ms/step - accuracy: 0.8200 - loss: 0.3903 - val_accuracy: 0.8242 - val_loss: 0.3831 - learning_rate: 5.0000e-04
Epoch 20/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 286ms/step - accuracy: 0.8056 - loss: 0.4056
```

10th to 20th epochs of model training

```
Epoch 20: val_loss did not improve from 0.37871
83/83 ━━━━━━━━━━━━━━━━━━━━ 30s 357ms/step - accuracy: 0.8056 - loss: 0.4055 - val_accuracy: 0.8167 - val_loss: 0.3826 - learning_rate: 5.0000e-04
Epoch 21/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 285ms/step - accuracy: 0.8169 - loss: 0.3880
Epoch 21: val_loss did not improve from 0.37871

Epoch 21: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
83/83 ━━━━━━━━━━━━━━━━━━━━ 29s 353ms/step - accuracy: 0.8168 - loss: 0.3881 - val_accuracy: 0.8227 - val_loss: 0.3792 - learning_rate: 5.0000e-04
Epoch 22/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 279ms/step - accuracy: 0.7919 - loss: 0.4171
Epoch 22: val_loss did not improve from 0.37871
83/83 ━━━━━━━━━━━━━━━━━━━━ 29s 346ms/step - accuracy: 0.7921 - loss: 0.4170 - val_accuracy: 0.8242 - val_loss: 0.3796 - learning_rate: 2.5000e-04
Epoch 23/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 293ms/step - accuracy: 0.8169 - loss: 0.3867
Epoch 23: val_loss did not improve from 0.37871
83/83 ━━━━━━━━━━━━━━━━━━━━ 30s 363ms/step - accuracy: 0.8169 - loss: 0.3869 - val_accuracy: 0.8288 - val_loss: 0.3805 - learning_rate: 2.5000e-04
Epoch 23: early stopping
Restoring model weights from the end of the best epoch: 18.
```

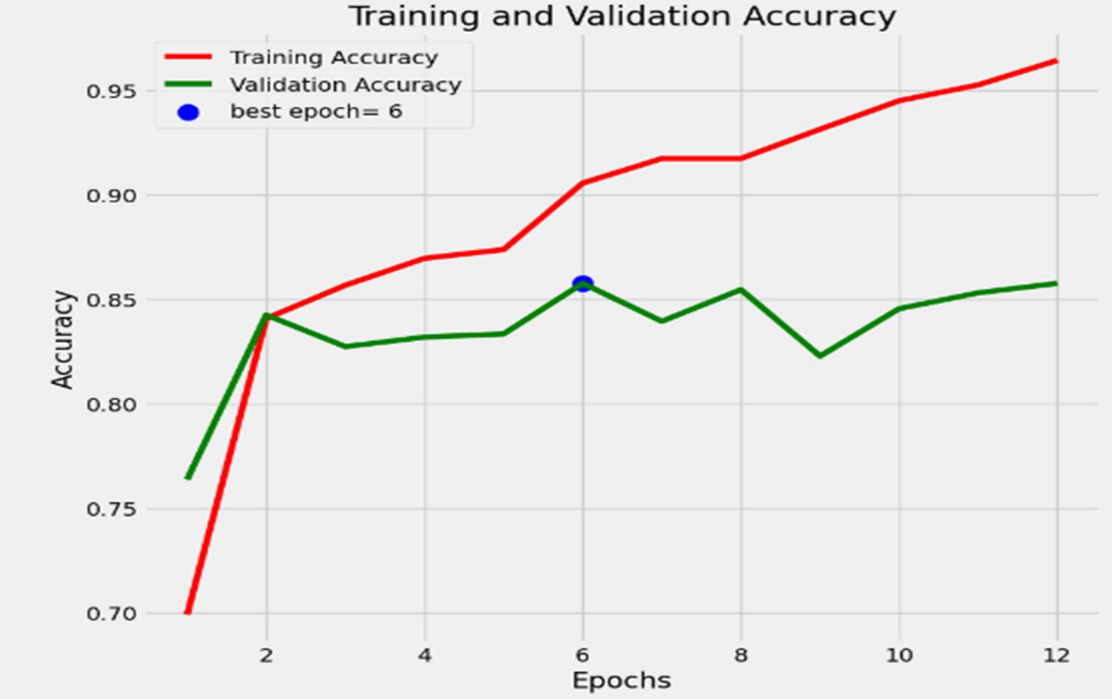
20th to 23th epochs of model training

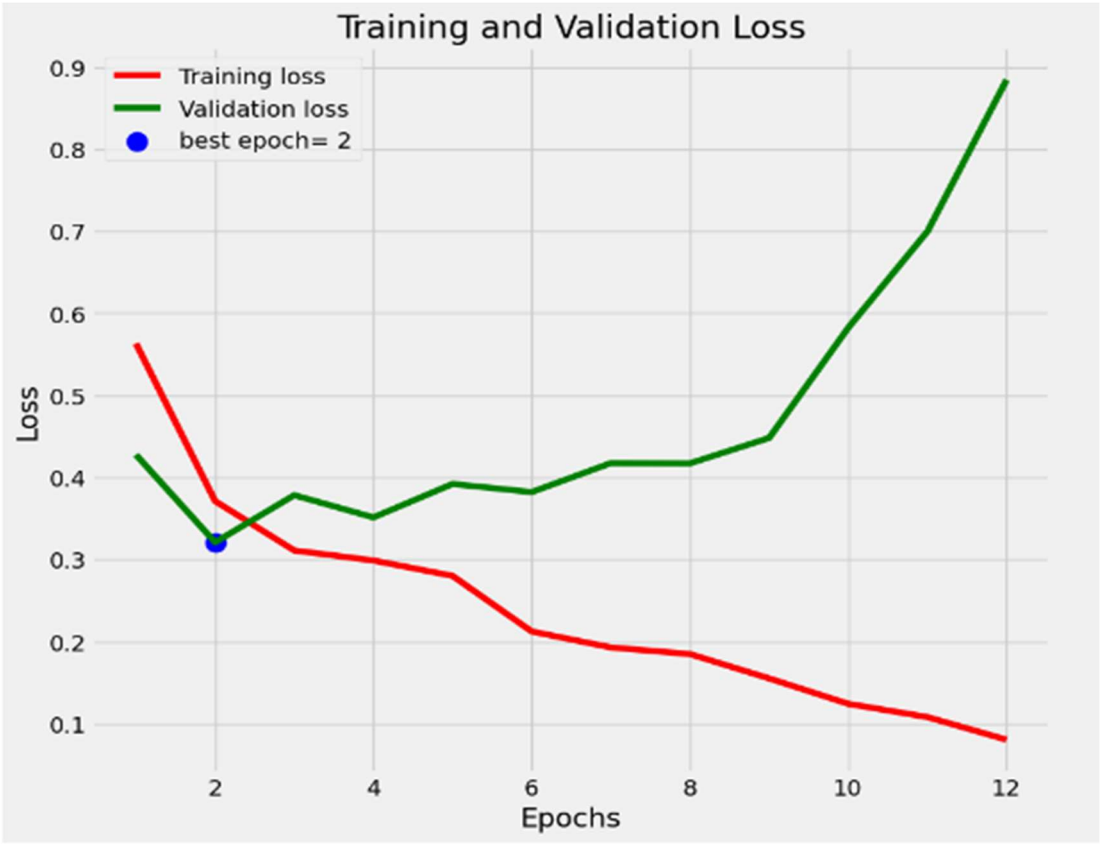
VGG16 Model:-

```
history = model.fit(X_train, y_train, batch_size=32,
                    callbacks=[early_stopping, model_checkpoint, lr_reduction],
                    validation_data=(X_test, y_test), epochs=25)
```

Training is performed based on different parameters like, epoch, batch size, learningrate etc.

- The number of epoch is set to 25 initially with a batch size of 32.
- Learning Rate : 0.00001
- The model gains an accuracy of 84.24% and a loss of 15.76%.






```
Epoch 1/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 701ms/step - accuracy: 0.6033 - loss: 0.6610
Epoch 1: val_loss improved from inf to 0.42763, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 80s 881ms/step - accuracy: 0.6045 - loss: 0.6598 - val_accuracy: 0.7636 - val_loss: 0.4276 - learning_rate: 0.0010
Epoch 2/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 679ms/step - accuracy: 0.8440 - loss: 0.3820
Epoch 2: val_loss improved from 0.42763 to 0.32094, saving model to best_model.keras
83/83 ━━━━━━━━━━━━━━━━━━━━ 69s 832ms/step - accuracy: 0.8440 - loss: 0.3818 - val_accuracy: 0.8424 - val_loss: 0.3209 - learning_rate: 0.0010
Epoch 3/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 686ms/step - accuracy: 0.8629 - loss: 0.3087
Epoch 3: val_loss did not improve from 0.32094
83/83 ━━━━━━━━━━━━━━━━━━━━ 68s 821ms/step - accuracy: 0.8628 - loss: 0.3087 - val_accuracy: 0.8273 - val_loss: 0.3784 - learning_rate: 0.0010
Epoch 4/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 688ms/step - accuracy: 0.8662 - loss: 0.3038
Epoch 4: val_loss did not improve from 0.32094
83/83 ━━━━━━━━━━━━━━━━━━━━ 68s 825ms/step - accuracy: 0.8662 - loss: 0.3037 - val_accuracy: 0.8318 - val_loss: 0.3513 - learning_rate: 0.0010
Epoch 5/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 678ms/step - accuracy: 0.8746 - loss: 0.2785
Epoch 5: val_loss did not improve from 0.32094

Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
83/83 ━━━━━━━━━━━━━━━━━━━━ 68s 815ms/step - accuracy: 0.8746 - loss: 0.2785 - val_accuracy: 0.8333 - val_loss: 0.3920 - learning_rate: 0.0010
Epoch 6/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 685ms/step - accuracy: 0.9095 - loss: 0.2063
Epoch 6: val_loss did not improve from 0.32094
83/83 ━━━━━━━━━━━━━━━━━━━━ 68s 820ms/step - accuracy: 0.9095 - loss: 0.2064 - val_accuracy: 0.8576 - val_loss: 0.3821 - learning_rate: 5.0000e-04
Epoch 7/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 669ms/step - accuracy: 0.9199 - loss: 0.1883
Epoch 7: val_loss did not improve from 0.32094
83/83 ━━━━━━━━━━━━━━━━━━━━ 66s 801ms/step - accuracy: 0.9199 - loss: 0.1884 - val_accuracy: 0.8394 - val_loss: 0.4174 - learning_rate: 5.0000e-04
Epoch 8/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 667ms/step - accuracy: 0.9169 - loss: 0.1814
Epoch 8: val_loss did not improve from 0.32094

Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
83/83 ━━━━━━━━━━━━━━━━━━━━ 66s 799ms/step - accuracy: 0.9169 - loss: 0.1815 - val_accuracy: 0.8545 - val_loss: 0.4171 - learning_rate: 5.0000e-04
```

First 8 epochs of model training

```
Epoch 9/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 674ms/step - accuracy: 0.9387 - loss: 0.1470
Epoch 9: val_loss did not improve from 0.32094
83/83 ━━━━━━━━━━━━━━━━━━━━ 67s 812ms/step - accuracy: 0.9386 - loss: 0.1471 - val_accuracy: 0.8227 - val_loss: 0.4480 - learning_rate: 2.5000e-04
Epoch 10/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 692ms/step - accuracy: 0.9452 - loss: 0.1198
Epoch 10: val_loss did not improve from 0.32094
83/83 ━━━━━━━━━━━━━━━━━━━━ 69s 832ms/step - accuracy: 0.9452 - loss: 0.1198 - val_accuracy: 0.8455 - val_loss: 0.5823 - learning_rate: 2.5000e-04
Epoch 11/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 691ms/step - accuracy: 0.9532 - loss: 0.1018
Epoch 11: val_loss did not improve from 0.32094

Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
83/83 ━━━━━━━━━━━━━━━━━━━━ 69s 830ms/step - accuracy: 0.9532 - loss: 0.1019 - val_accuracy: 0.8530 - val_loss: 0.6996 - learning_rate: 2.5000e-04
Epoch 12/25
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9635 - loss: 0.0840
Epoch 12: val_loss did not improve from 0.32094
83/83 ━━━━━━━━━━━━━━━━━━━━ 222s 3s/step - accuracy: 0.9635 - loss: 0.0839 - val_accuracy: 0.8576 - val_loss: 0.8843 - learning_rate: 1.2500e-04
Epoch 12: early stopping
Restoring model weights from the end of the best epoch: 2.
Accuracy: 84.24%
Error: 15.76%
```

20th to 23th epochs of model training

4.1.3. Testing

Testing or validation data is used to evaluate our model‘s accuracy. To check whetherthe application is able to predict the output correctly. 30% of the dataset are used for testing the model.

```
Epoch 31/50
        222s 3s/step -
        loss: 0.0839 -
        accuracy: 0.9635 -
        val_loss: 0.8843 -
        val_accuracy: 0.8676
```

The models are tested along with training. At each epoch the model tries to predict the unseen test data. The accuracy on the test dataset changes with each epoch and the best model with maximum accuracy and minimum loss is selected. In the 12th epoch, the validation accuracy is 84.24%.

5. RESULTS AND DISCUSSION

The aim of this project is to develop a robust system for detecting skin cancer using advanced deep learning techniques. The system leverages a custom convolutional neural network (CNN) model, alongside pretrained models such as ResNet-50 and VGG16, to enhance the accuracy and reliability of skin lesion classification.

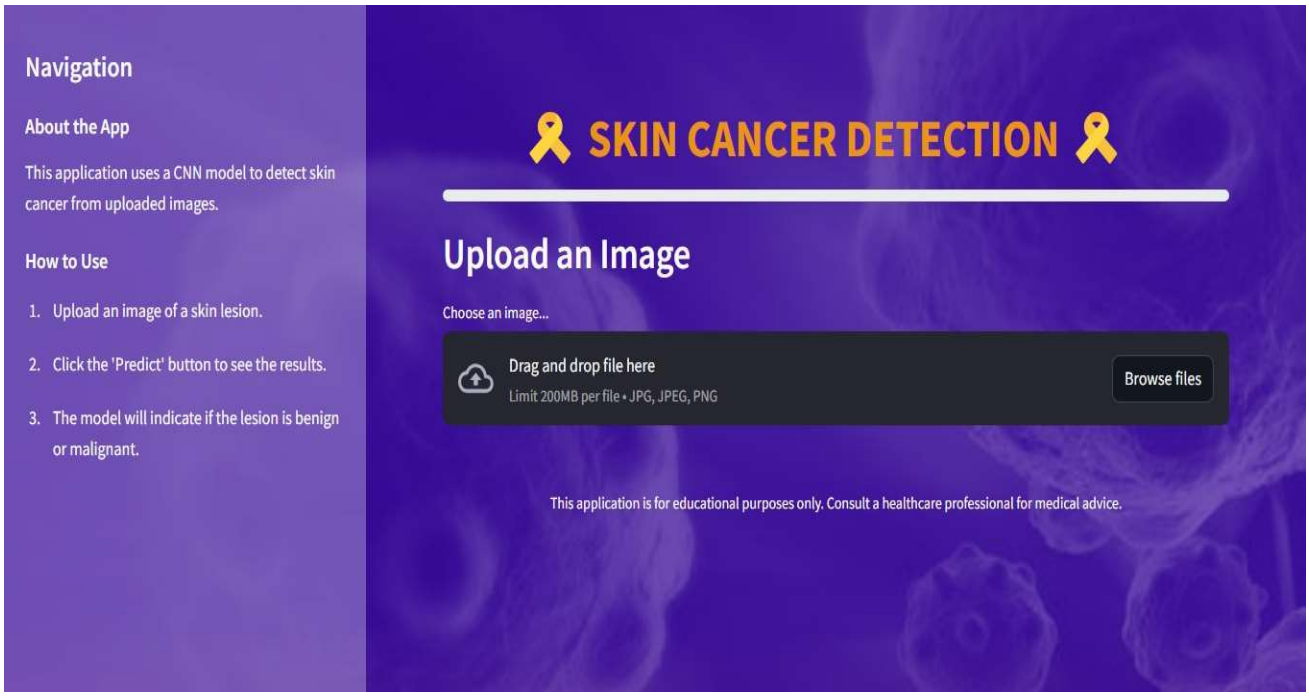
Given the variability in skin lesion appearances, including differences in size, color, and texture, the system is designed to effectively process a diverse dataset. Each model is fine-tuned to adapt to the unique characteristics of the data, ensuring high performance across various skin types and lesion conditions. Accuracy is considered as the metrics to measure the effectiveness of the system. It is also used in the training of the dataset. Accuracy is a measurement of observational error. It defines how close or far off a given set of measurement are to their true value. Out of the total 30 epochs, I selected the model saved at the 12^{th} epoch with validation accuracy 84.24% and minimum validation loss 15.76%. This shows that even at high validation accuracy the model may not provide the minimum validation loss.

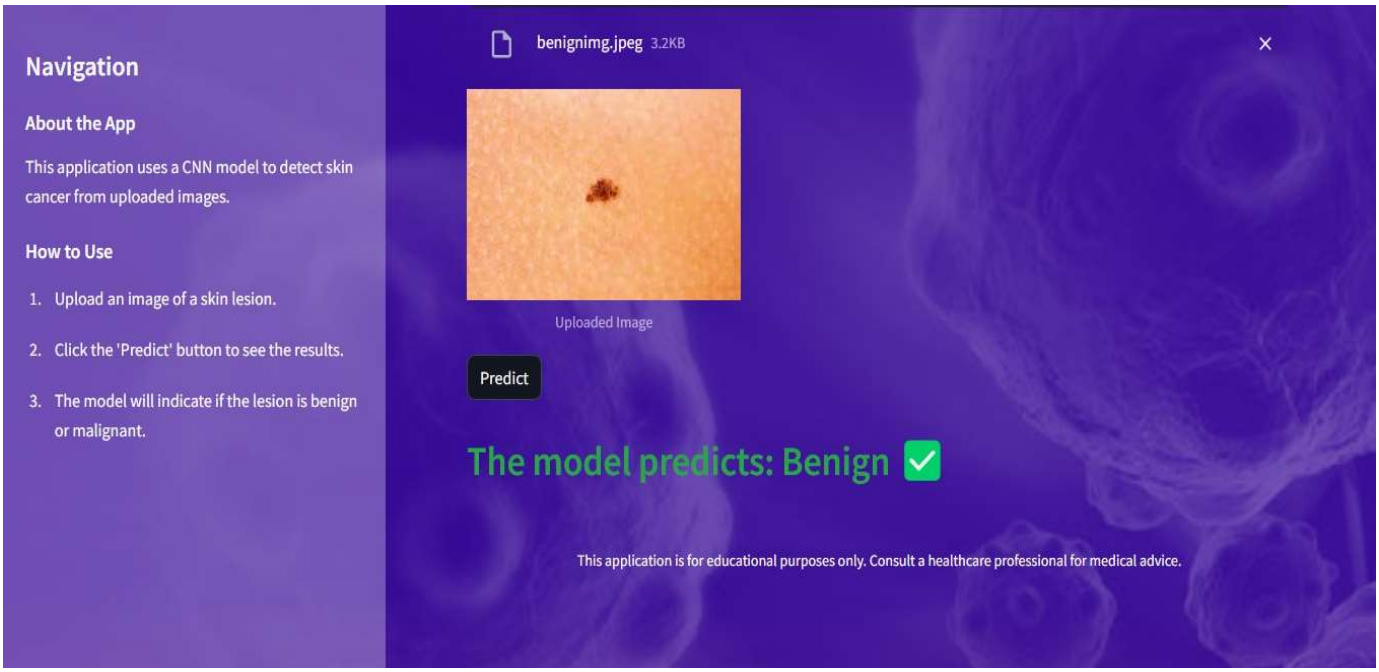
The project highlights the importance of extensive training data in improving predictive performance. The results suggest that with sufficient training and model optimization, our system can achieve high accuracy in skin cancer detection, potentially aiding in early diagnosis and treatment.

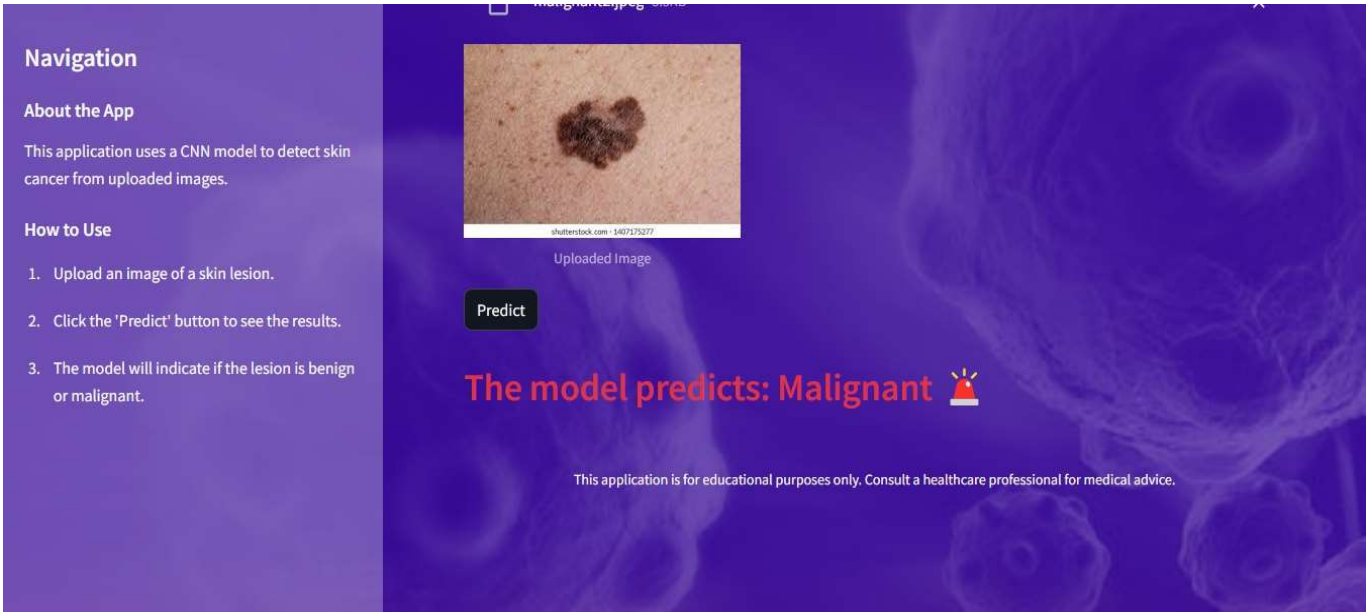
6. MODEL DEPLOYMENT

This figure shows the user interface of this application. The interface is very simple and easy to understand. There are only some elements displayed on the screen. There is a file upload option provided. The user can choose image from the local storage. The prediction will be initiated when the user submits that form, the image uploaded will be used for analysis. The output will be whether the lesions are benign or malignant.

UI DESIGN

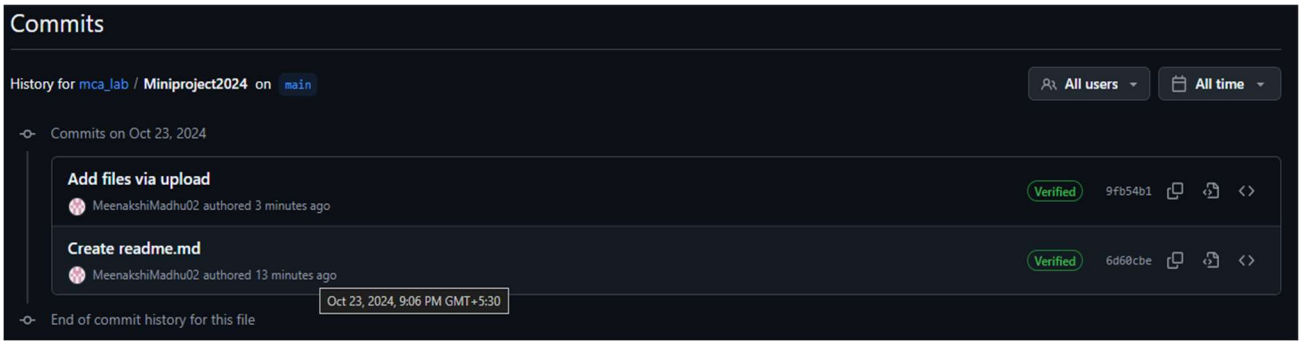



Navigation
About the App
This application uses a CNN model to detect skin cancer from uploaded images.
How to Use
1. Upload an image of a skin lesion.
2. Click the 'Predict' button to see the results.
3. The model will indicate if the lesion is benign or malignant.
benignimg.jpeg 3.2KB
Uploaded Image
Predict
The model predicts: Benign
This application is for educational purposes only. Consult a healthcare professional for medical advice.


Navigation
About the App
This application uses a CNN model to detect skin cancer from uploaded images.
How to Use
1. Upload an image of a skin lesion.
2. Click the 'Predict' button to see the results.
3. The model will indicate if the lesion is benign or malignant.
Uploaded Image
Predict
The model predicts: Malignant
This application is for educational purposes only. Consult a healthcare professional for medical advice.

7. GIT HISTORY

https://github.com/MeenakshiMadhu02/mca_lab/tree/main/Miniproject2024



8. CONCLUSION

As skin cancer continues to be a significant health concern, affecting millions worldwide, early detection is crucial for improving outcomes and saving lives. This system aims to provide an accessible solution for identifying potential skin lesions through advanced image processing and machine learning techniques.

This is an image processing application that analyzes skin lesion images to determine whether they are benign or malignant. The system is developed using deep learning techniques, incorporating models such as ResNet 50, VGG16, and a custom CNN. These models are widely recognized for their effectiveness in image classification tasks, providing a robust solution for skin cancer detection.

Considering the critical need for accurate skin cancer detection, this project develops an efficient system based on ResNet 50, VGG16, and a custom CNN to analyze skin lesion images. The system utilizes a dataset that includes diverse skin lesion images collected under various lighting conditions, skin tones, and lesion types for training. This comprehensive approach enhances the model's accuracy and reliability in diagnosing skin cancer across different scenarios.

The model developed demonstrates an accuracy of 84.24% in identifying skin cancer from images of skin lesions.The system is capable of accurately detecting lesions that are clearly visible as well as subtler abnormalities, enhancing its effectiveness in early skin cancer detection.

The models were trained three times, each with a different architecture—one custom CNN model, ResNet50, and VGG16—varying the number of epochs in each iteration. An optimal count of forty epochs was selected for the final training phase across all models. By saving the model after each epoch, we can identify the best-performing model based on validation loss and validation accuracy, ensuring the highest level of reliability in skin cancer detection.

9. FUTURE WORK

The model is developed as a web-based application that can detect skin cancer from uploaded images of skin lesions. This detection technique can be integrated into telehealth platforms, allowing healthcare providers to evaluate skin lesions remotely and make informed decisions. Additionally, the application can be used in mobile health applications to alert users about potential skin issues, encouraging early consultations with dermatologists. By providing real-time analysis and feedback, the system aims to promote awareness about skin health and facilitate timely interventions.

Another application is that, this technology could be deployed in community health initiatives, where public health organizations utilize the data collected to identify regions with a higher incidence of skin conditions. This information could guide targeted awareness campaigns and screening programs, ultimately fostering early detection and improved public health outcomes.

10. APPENDIX

10.1. Minimum Software Requirements

Operating System: Windows, Linux , Jupyter Notebook

10.2. Minimum Hardware Requirements

Hardware capacity	:	256 GB
(minimum)RAM	:	8 GB
Processor	:	Intel Core i3
GPU:	:	2 GB
Display	:	1366 * 768

11. REFERENCES

- Malo, Dipu Chandra, et al. "Skin cancer detection using convolutional neural network." 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2022.
- Sharma, Deepti, and Swati Srivastava. "Automatically detection of skin cancer by classification of neural network." International Journal of Engineering and Technical Research 4.1 (2016): 15-18.