A Project Report

on

Skin Lesion Classification

Submitted in partial fulfilment of the requirements for the award of degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

by

17WH1A0522	Ms. K HARSHINI
17WH1A0543	Ms. V PRAVALIKA
17WH1A0509	Ms. K CHAITRA

Under the esteemed guidance of

Mr. Y MADAN REDDY

Assistant Professor

Department of Computer Science and Engineering BVRIT HYDERABAD

College of Engineering for Women

(NBA Accredited EEE, ECE, CSE, IT B.Tech. Courses,

Accredited by NAAC with „A" Grade)

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

May 2021

BVRIT HYDERABAD College of Engineering for Women

(NBA Accredited EEE, ECE, CSE, IT B.Tech. Courses,

Accredited by NAAC with „A" Grade)

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project work report entitled "**Skin Lesion Classification**" is a bonafide work carried out by **Ms. K. Harshini (17WH1A0522), Ms. V. Pravalika (17WH1A0543), Ms. K. Chaitra(17WH1A0509)** in partial fulfillment for the award of B.Tech degree in **Computer Science & Engineering**, **BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide   **Head of the Department**

Mr. Y Madan Reddy   **Dr. K. Srinivasa Reddy**

Assistant Professor   Professor & HOD

Dept of CSE   **Dept of CSE**

External Examiner

DECLARATION

We hereby declare that the work presented in this project entitled **"SKIN LESION CLASSIFICATION"** submitted towards completion of Project Work in IV year of B.Tech., CSE at 'BVRIT HYDERABAD College of Engineering For Women', Hyderabad is an authentic record of our original work carried out under the guidance of Mr. Y.Madan Reddy, Assistant Professor, Department of CSE.

Sign. with date:

Ms. K.HARSHINI (17WH1A0522)

Sign. with date:

Ms. V.PRAVALIKA (17WH1A0543)

Sign. with date:

Ms. K.CHAITRA (17WH1A0509)

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal**, **BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. K.Srinivasa Reddy, Professor**, Department of CSE, **BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Mr. Y.Madan Reddy, Assistant Professor**, Department of CSE**, BVRIT HYDERABAD College of Engineering for Women** for his constant guidance, encouragement, and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of the **CSE** Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Ms. K.HARSHINI (17WH1A0522)

Ms. V.PRAVALIKA (17WH1A0543)

Ms. K.CHAITRA (17WH1A0509)

Contents

ABSTRACT

Melanoma is deadly skin cancer. There is a high similarity between different kinds of skin lesions, which leads to incorrect classification. Accurate classification of a skin lesion in its early stages saves human life. This project aims at classifying skin lesion images into seven classes namely melanoma, melanocytic nevus, basal cell carcinoma, actinic keratosis, benign keratosis, dermatofibroma, and vascular lesion using deep convolutional neural networks (CNNs) with mobilenet architecture. In this, a lesion image from the dataset is taken as input and is expected to give the type of lesion as output.

LIST OF FIGURES

1. INTRODUCTION

Skin cancer, a major form of cancer, is a critical public health problem with 123,000 newly diagnosed melanoma cases and between 2 and 3 million non-melanoma cases worldwide each year. The leading cause of skin cancer is high exposure of skin cells to UV radiation, which can damage the DNA inside skin cells leading to uncontrolled growth of skin cells. Skin cancer is primarily diagnosed visually employing clinical screening, a biopsy, dermoscopic analysis, and histopathological examination. It has been demonstrated that the dermoscopic analysis in the hands of inexperienced dermatologists may cause a reduction in diagnostic accuracy. This project aims at classifying skin lesion images into seven classes namely melanoma, melanocytic nevus, basal cell carcinoma, actinic keratosis, benign keratosis, dermatofibroma, and vascular lesion using deep convolutional neural networks (CNNs) with mobilenet architecture. A CNN tries to mimic the process of recognition of images by the visual cortex in the brain. For better results in image classification, feature extraction is used according to machine learning tasks. In CNN architecture there are multiple convolution layers followed by a stack of pooling layers, then contrast normalization layers. In the end, there is at least one fully connected layer. A CNN is less dense and can easily be trained compared to the feed-forward network. However, the best performance of CNNs is slightly lower and more computationally heavy on high-resolution images than the feed-forward network.

1.1 Objectives

Identifying skin lesions is time-consuming and often leads to incorrect classification. This sometimes leads to the late discovery of deadly skin cancer. Hence, accurate and fast classification of a skin lesion in its early stages can save someone's life. Skin lesion classification aims at identifying different kinds of skin lesions anywhere on a person's body. Skin cancer is a major threat to humans and can sometimes lead to death, but their quick identification remains difficult. Using the HAM10000 dataset containing seven types of lesion classifications, we train a deep convolutional neural network to identify the type of skin lesion.

1.2 Methodology

To classify lesions, a large collection of skin lesion images on different parts of the body is required. The dataset is downloaded from Kaggle. In this section the methodology followed is discussed in detail.

1.2.1 Dataset

The dataset for the experiment is downloaded from Kaggle which contains different skin lesion images and their labels. It contains a collection of images taken from different people, present on different parts of their bodies. The dataset consists of a total of 10015 dermoscopy images, which are released as a training set for academic machine learning purposes and are publicly available through the ISIC archive. It includes 6705 Melanocytic nevi images, 1113 Melanoma images, 1099 Benign keratosis images, 514 Basal cell carcinoma images, 327 Actinic keratosis images, 142 Vascular images, and 115 Dermatofibroma images with 600 x 450 pixels resolution.
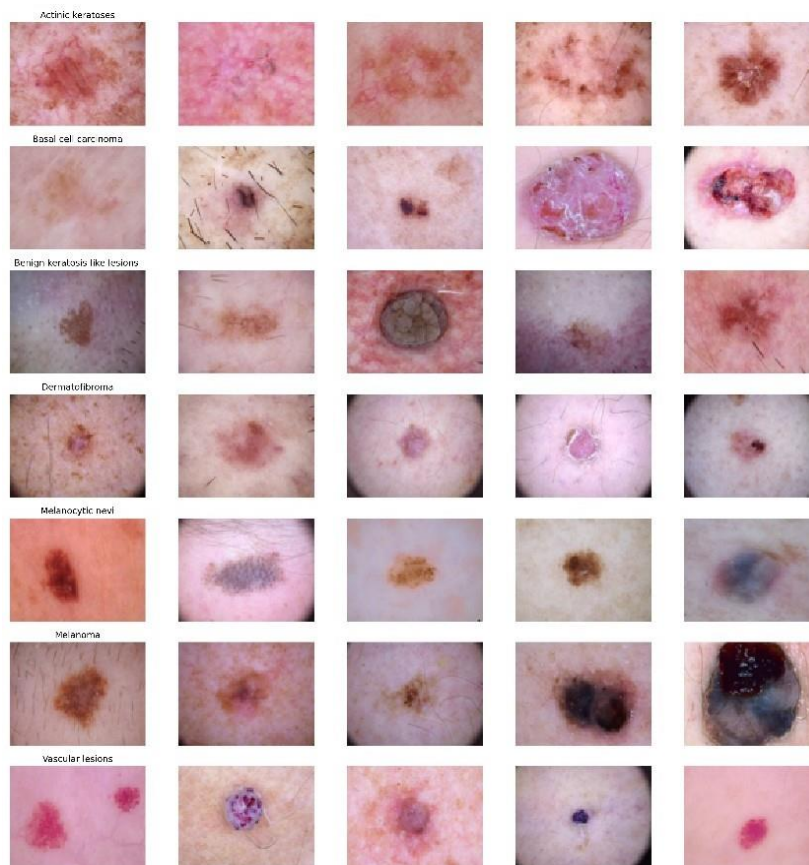
Fig. 1.2.1 Sample images from HAM10000 dataset for cancer types (a)Actinic Keratosis (b)Basal Cell Carcinoma (c)Benign Keratosis (d)Dermatofibroma (e)Melanocytic nevi (f)Melanoma (g)Vascular Lesions

The dataset includes lesions with multiple images, which can be tracked by the lesion_id column within the HAM10000_metadata file. Which contains 7 attributes namely lesion_id, image_id, dx, dx_type, age, sex, localization.

Fig. 1.2.2 Metadata file

Data pre-processing and cleaning

We found there were 57 null age entries in the dataset which were filled using the mean filling method. The images were resized into 224 x 224 as the original dimension of images is 450 x 600 cannot be handled by TensorFlow.

Fig.1.2.1.2 Data preprocessing and cleaning

Data Augmentation

This step is required to avoid the overfitting problem. For this, we need to expand our HAM 10000 dataset artificially. Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as data augmentation. For the data augmentation, random rotation of some training images by 10 degrees, random zooming of some training images by 10%, random shifting of images horizontally by 10% of the width, random shifting of images vertically by 10% of the height were implemented.

Train-Test Split

The 10015 images in the dataset were split into training and testing sets of 80:20 ratio. Then the train set was split into two parts: a small fraction (10%) became the validation set which the model evaluated and the rest (90%) is used to train the model.

Fitting the model

In this step, the model is fit into x_train, y_train. Then they are sent to fit_generator function with optimal batch_size and epochs required to train the model sufficiently.

1.2.2 The proposed CNN model with MobileNet architecture

We initially implemented a sequential CNN model with a few of 12 layers being 4 Conv2D layers, 2 MaxPooling2D layers, 3 Dropout, Flatten, and Dense layers. But later changed it to MobileNet architecture for better accuracy. MobileNet in total consists of 25 layers, which employs 4 Conv2D layers, 7 BatchNormalization layers,

7 ReLU layers, 3 ZeroPadding2D layers, and 1 DepthwiseConv2D, GlobalAveragePooling, Dropout, and Dense layers. The whole model is built from carefully repeating a set of MobileNet blocks with different filter and strides values.
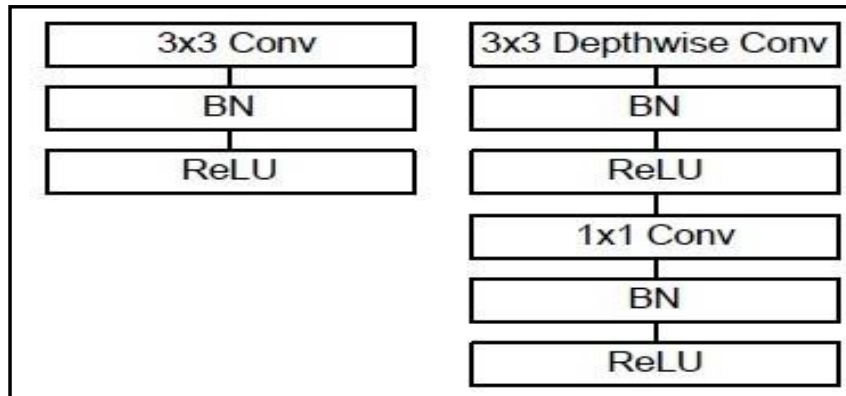
Fig.1.2.2.1 MobileNet blocks

For MobileNets the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs of the depthwise convolution. A standard convolution, both, filters and combines inputs into a new set of outputs in one step. The depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size.
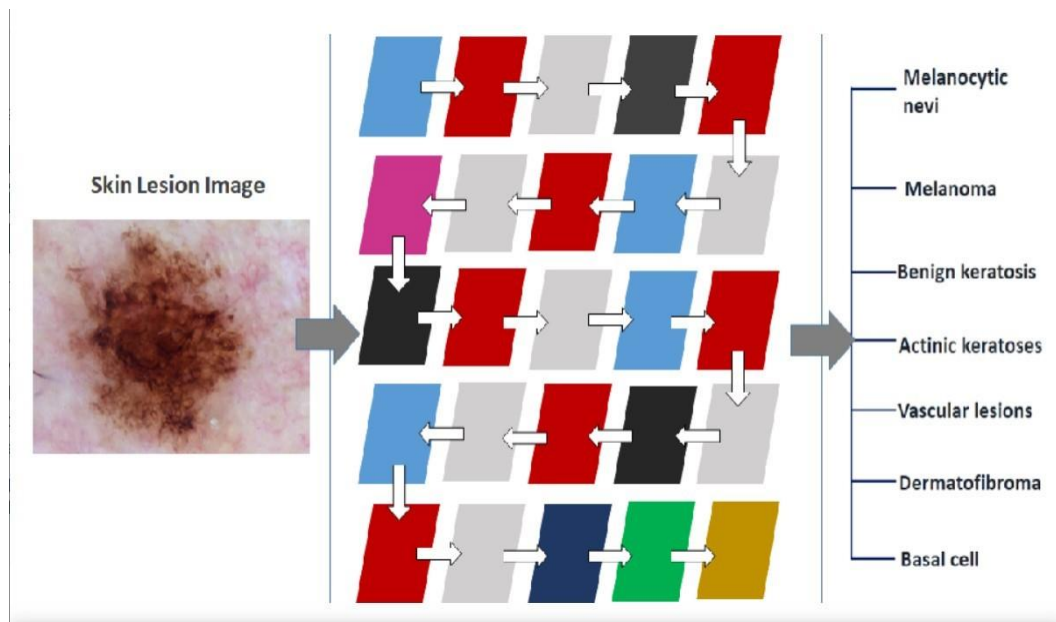
Fig.1.2.2.2 MobileNet Architecture

Main layers of MobileNet architecture

Convolution: The term convolution refers to the mathematical combination of two functions to produce a third function. In the case of a CNN, the convolution is performed on the input data with the use of a filter to then produce a feature map.

Depthwise Convolution: Depthwise convolution is a type of convolution in which a single convolutional filter is applied to each input channel (i.e. in a depthwise way). You can understand depthwise convolution as being the first step in a depthwise separable convolution.

It is implemented via the following steps:

Split the input into individual channels.

Convolve each input with the layer's kernel (called a depthwise kernel).

Stack the convolved outputs together (along the channels axis).

(a) Standard Convolution Filters

(b) Depthwise Convolutional Filters

(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution
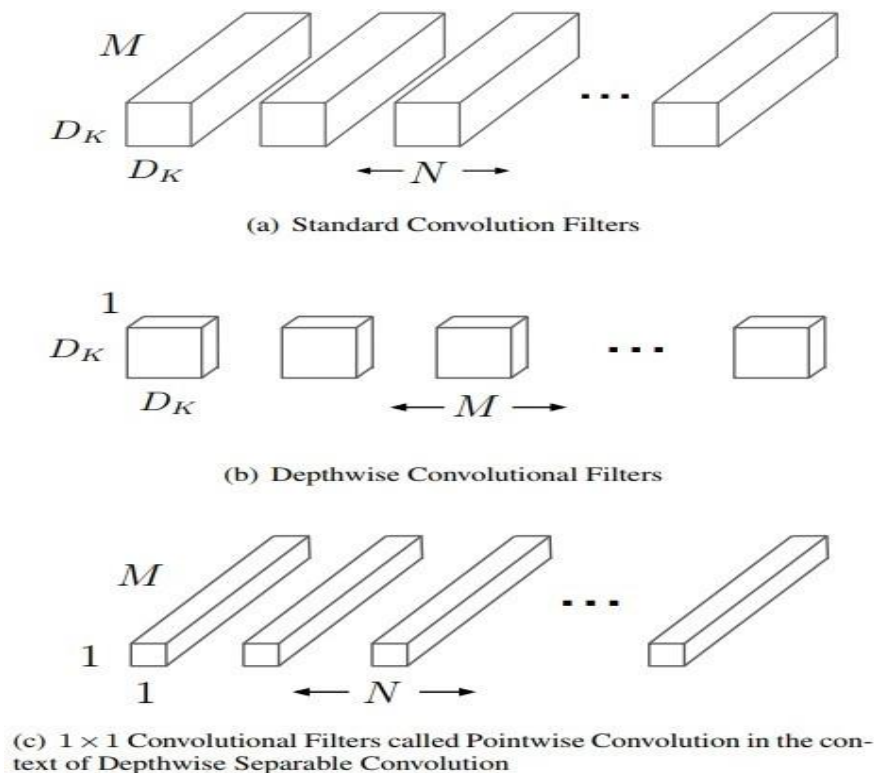
Fig.1.2.2.3 Depthwise vs standard convolution layers

Batch Normalisation: Layer that normalizes its inputs. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. Importantly, batch normalization works differently during training and inference.

During training, the layer normalizes its output using the mean and standard deviation of the current batch of inputs. **During inference,** the layer normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training.
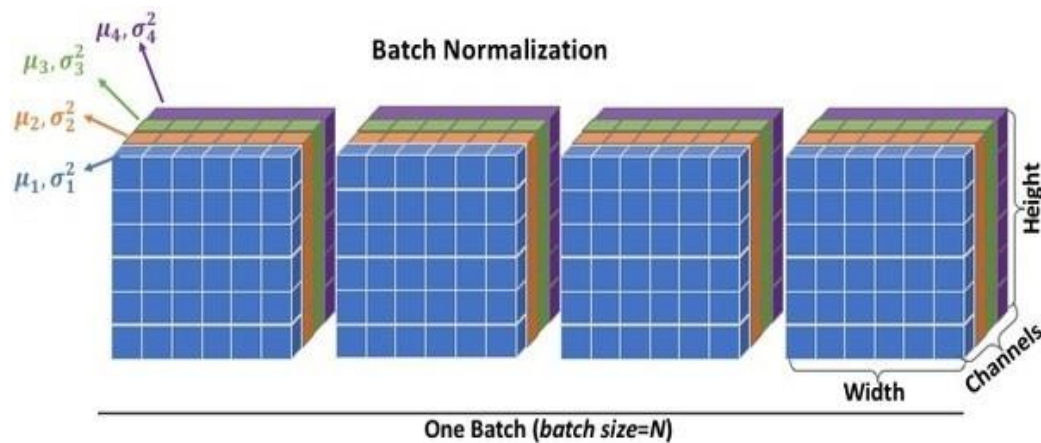
Fig.1.2.2.4 Batch Normalisation layer

Applying ReLu(Rectified Linear Unit): In this step, we apply the rectifier function to increase non-linearity in the CNN. Images are made of different objects that are not linear to each other. Without applying this function the image classification will be treated as a linear problem while it is actually a non-linear one.

Max pooling: Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned**.**
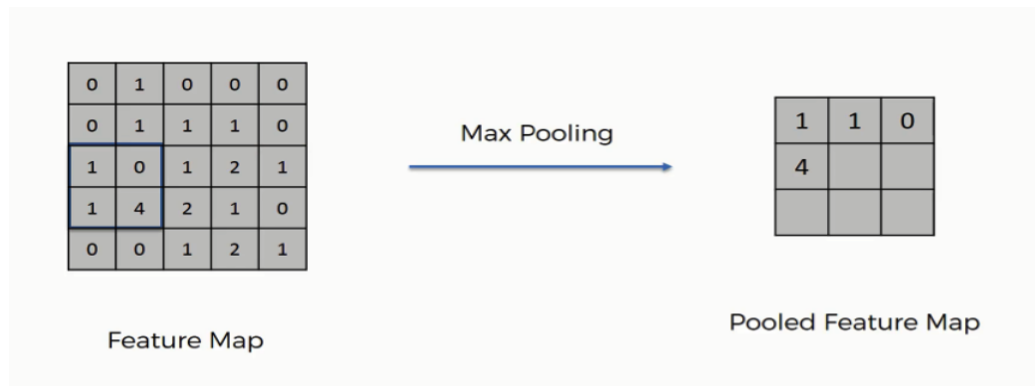
Fig.1.2.2.5 MaxPooling layer

Flattening: Once the pooled featured map is obtained, the next step is to flatten it. Flattening involves transforming the entire pooled feature map matrix into a single column which is then fed to the neural network for processing. Flattening is also known as the process of converting all the resultant 2-dimensional arrays into a single long continuous linear vector.
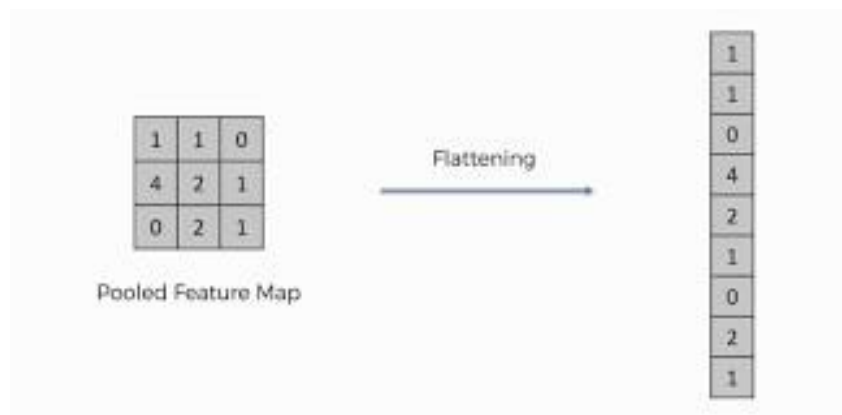
Fig.1.2.2.6 Flattening layer

Dropout: Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.

During training, some number of layer outputs are randomly ignored or "*dropped out.*" This

has the effect of making the layer look-like and be treated like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "*view*" of the configured layer. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections

Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsible for the inputs.

Dense: The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.

In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.

The output generated by the dense layer is an 'm' dimensional vector. Thus, the dense layer is used for changing the dimensions of the vector. Dense layers also apply operations like rotation, scaling, translation on the vector.

Full Connection: At the end of a CNN, the output of the last Pooling Layer acts as an input to the so-called Fully Connected Layer. There can be one or more of these layers ("fully connected" means that every node in the first layer is connected to every node in the second layer).

1.3 Organization of Project

For this project, a lesion image from the dataset is taken as input and expected to give the type of lesion using a convolutional neural network. According to the input image, lesion name/type, its index number path of the image is given.

2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

2.1 Requirements Gathering

2.1.1 Software Requirements

Programming Language: Python 3.6+

Graphical User Interface: Tkinter

Dataset: HAM10000 Dataset

Packages: Numpy, Pandas, Matplotlib, Scikit-learn, Keras, Tensorflow

IDE: Google Colab

2.1.2 Hardware Requirements

Operating System: Windows 7+,

Ubuntu Processor: Intel Core i3-2348M

CPU Speed: 2.30 GHz

RAM: 4 GB

2.2 Technologies Description

Python

Python is an interpreted high-level general-purpose programming language, has a design philosophy that emphasizes code readability with its notable use of significant indentation, the recommended indent size is four spaces. Developed by Guido van Rossum in the late 1980s and first released in 1991. Python strives for a simpler, less-cluttered syntax and grammar. Python is used for developing web applications, data science, and rapid application development. Python is portable, extensible, embeddable, and scalable.

Tensorflow

TensorFlow is a free and open-source software library for machine learning, based on data flow and differentiable programming across a range of tasks. It is a symbolic math library. It is also used for machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow accepts data in the form of multi-dimensional arrays of higher dimensions called tensors.

It was developed by the Google Brain team for internal Google use and was released under the Apache License 2.0 in 2015. Google released Collaboratory, which is a TensorFlow Jupyter notebook environment that requires no setup to use.

Keras:

Keras is an open-source software library that provides a Python interface for artificial neural networks. It is a deep learning API. It runs on top of the machine learning platform Tensorflow. It is user-friendly, modular, and extensible. It acts as an interface for the TensorFlow library.

The core data structures of Keras are layers and models. It contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object

- Sophisticated (broadcasting) functions

- Tools for integrating C/C++ and Fortran code

- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python library providing high-performance data manipulation and analysis tools using its powerful data structures. Python was majorly used for data and preparation. It had very little contribution to data analysis.

Pandas solved this problem.   Using Pandas,   we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, statistics, analytics, etc.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn.

Google Colaboratory

Collaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser and is especially well suited to machine learning, data analysis, and education.
Colab is a free Jupyter notebook environment that runs entirely in the cloud.

Google Colab allows to:

Write and execute code in Python

Document your code that supports mathematical equations

Create/Upload/Share notebooks

Import/Save notebooks from/to Google Drive

Import/Publish notebooks from GitHub

Import external datasets e.g. from Kaggle

Integrate PyTorch, TensorFlow, Keras, OpenCV

Colab works with most major browsers and is most thoroughly tested with the latest versions of Chrome, Firefox, and Safari.

Tkinter

Tkinter is the de facto way in Python to create <u>Graphical User interfaces</u> and is included in all standard Python Distributions. This Python framework provides an interface to the Tk toolkit and works as a thin object-oriented layer on top of Tk. The Tk toolkit is a cross-platform collection of 'graphical control elements', aka widgets, for building application interfaces.

This framework provides Python users with a simple way to create GUI elements using the widgets found in the Tk toolkit. Tk widgets can be used to construct buttons, menus, data fields, etc. in a Python application. Once created, these graphical elements can be associated with or interact with features, functionality, methods, data or even other widgets.

PIL Pillow (Python Imaging Library)

The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

3. DESIGN

3.1 Introduction

Software design sits at the technical kernel of the software engineering process and is applied regardless of the developmental paradigm and area of application. Design is the first step in the developmental phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirements have been specified and analyzed, system design is the first of the three technical activities – design, code, and test.

The importance can be stated with the single word "Quality". Design is the place where quality is fostered in software development. The design provides us with representations of software that can assess for quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design, we risk building an unstable system – one that will be difficult to test, one whose quality cannot be assessed until the last stage. During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed, and documented. System design can be viewed from either a technical or project management perspective. From the technical point of view, design is comprised of four activities – architectural design, data structure design, interface design, and procedural design.

3.2 Architecture Diagram

Web applications are by nature distributed applications, meaning that they are programs that run on more than one computer and communicate through a network or server; specifically, web applications are accessed with a web browser and are popular because of the ease of using the browser as a user client. For the enterprise, software on potentially thousands of client computers is a key reason for their popularity. Web applications are used for webmail, online retail sales, discussion boards, weblogs, online banking, etc. One web application can be accessed and used by millions of people.
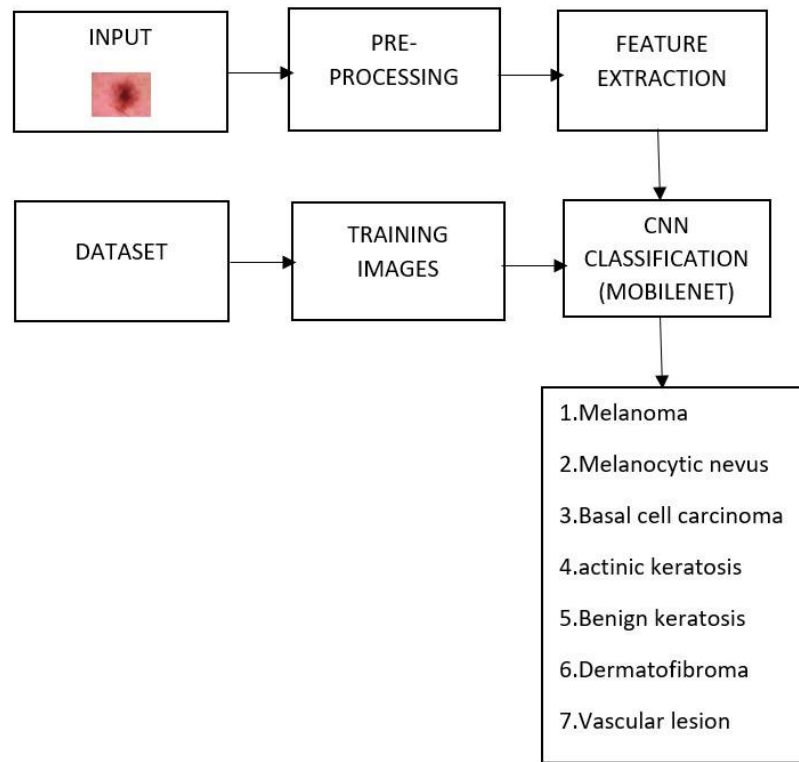
Fig.3.2 Architecture Diagram

3.3 UML Diagrams

3.3.1 Use Case Diagram

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.

Only static behavior is not sufficient to model a system rather dynamic behavior is more than static behavior. In UML, there are five diagrams available to model the dynamic nature, and a use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic, there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. Use case diagrams consist of actors, use cases, and their relationships. The diagram is used to model the system/subsystem of an application. A single-use case diagram captures a particular functionality of a system.
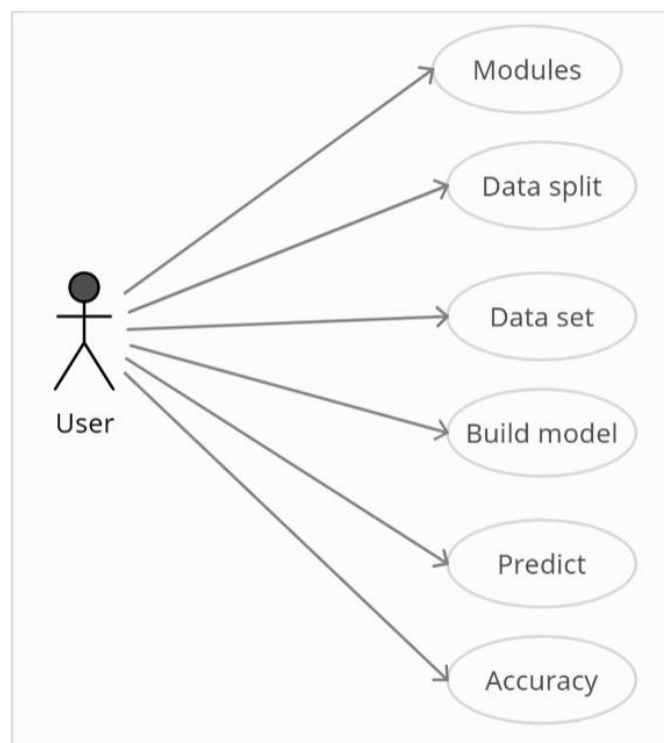
Fig.3.3.1 Usecase Diagram

3.3.2 Sequence Diagram

The sequence Diagram represents the objects that participate in the interactions horizontally and time vertically. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.
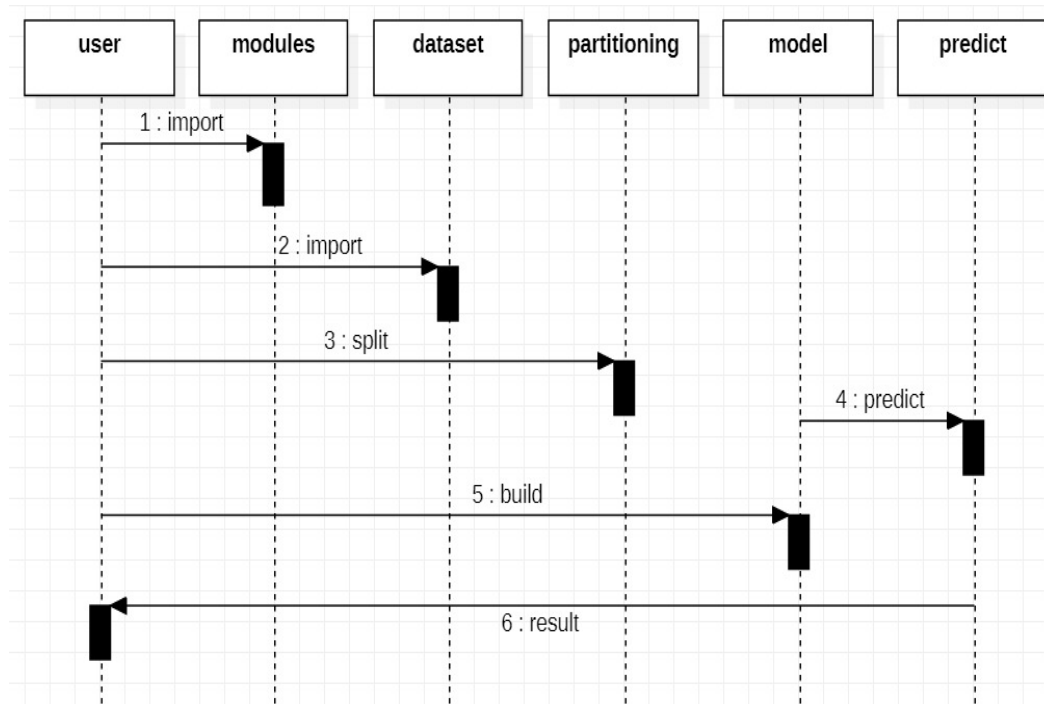
Fig.3.3.2 Sequence Diagram

3.3.4 Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.
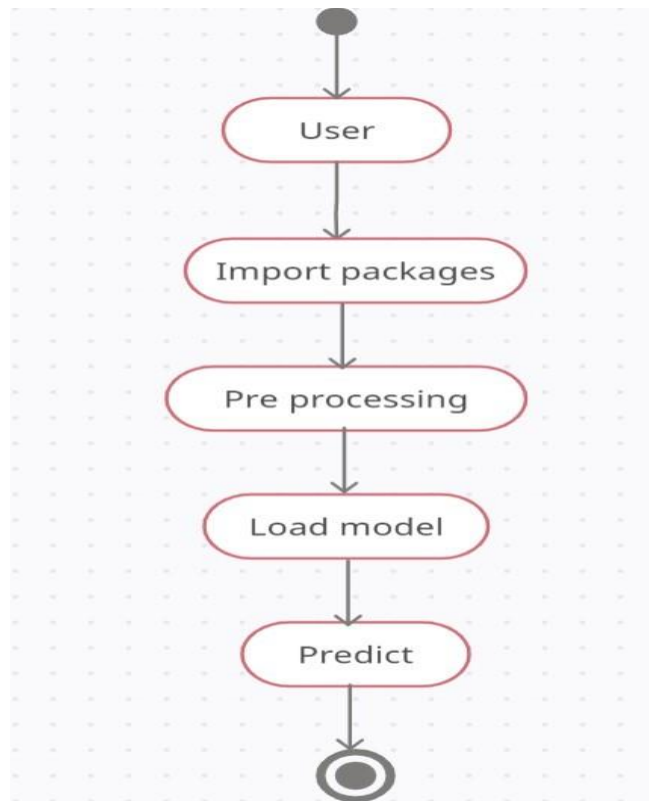
Fig3.3.3 Activity Diagram

3.3.5 Collaboration Diagram

A collaboration diagram resembles a flowchart that portrays the roles, functionality, and behavior of individual objects as well as the overall operation of the system in real-time. Objects are shown as rectangles with naming labels inside. These labels are preceded by colons and may be underlined. The relationships between objects are shown as arrows connecting the relevant rectangles along with labels that define the message sequencing.
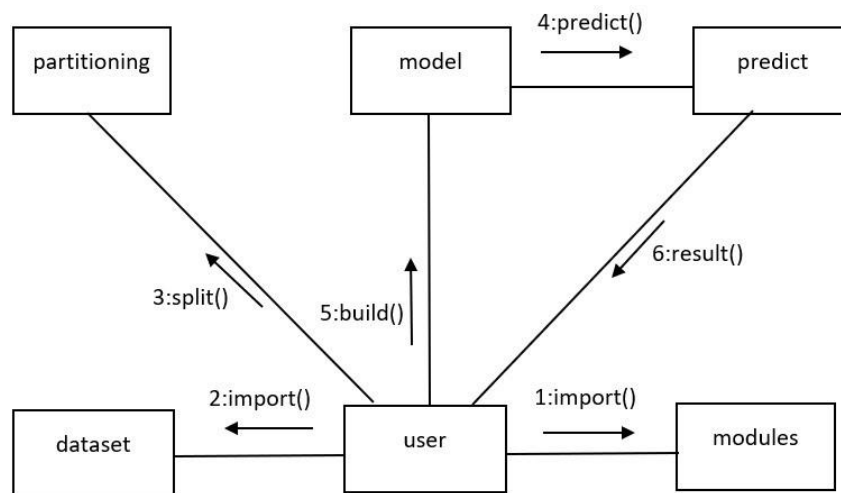
Fig.3.3.4 Collaboration Diagram

3.3.6 Class Diagram

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the systematics of the application, and the detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the applications, and the classes to be programmed.
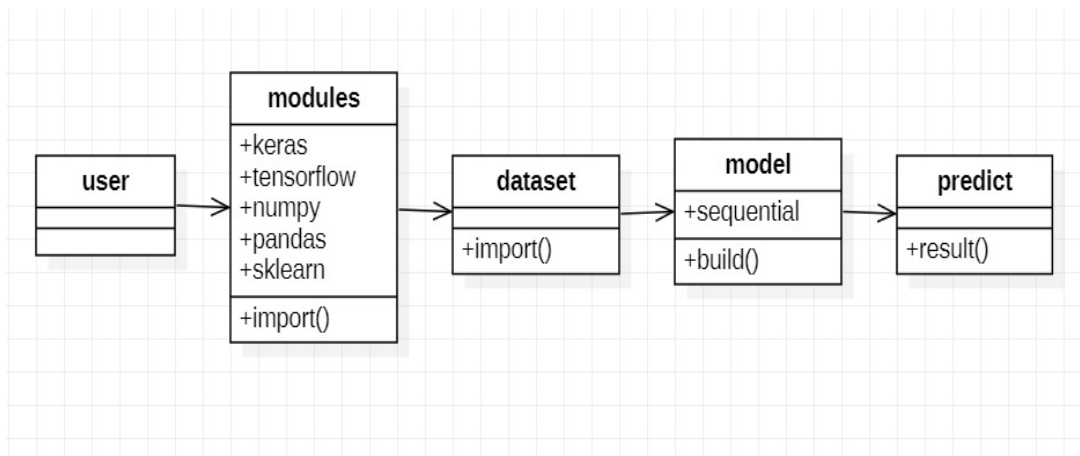
Fig.3.3.5 Class Diagram

3.4 Phases of project
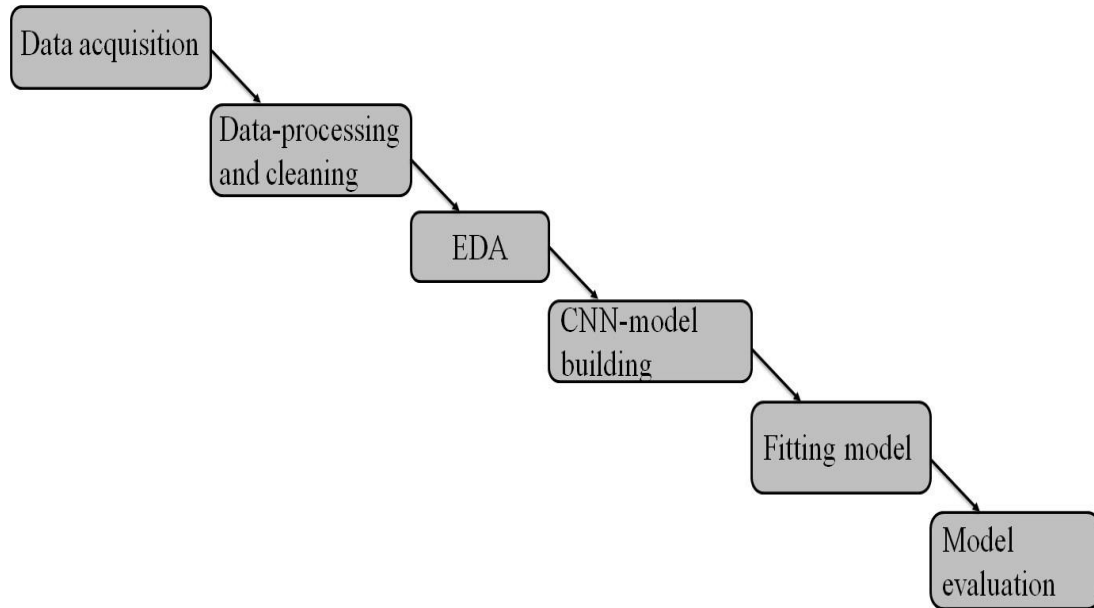
Fig.3.4.1 Phases

The first and foremost step is data acquisition, where we took the dataset "HAM10000" from Kaggle as discussed previously.

Next is a data processing and cleaning, where we tried finding any anomalies in the dataset and tried correcting them, for example, the age column had 57 null values which we replaced by the column mean value.

Next is EDA - Exploratory Data Analysis, here we tried analyzing our dataset using visualization tools like matplotlib. This step aimed to understand our dataset in a more statistical way.

Next is CNN model building. Initially, we built a regular sequential model with a few of 12 layers but later changed it to mobilenet architecture comprising of 25 layers.

Next is fitting the model. This step was performed to prevent the model from overfitting we have implemented LRAnnealation and Data Augmentation.

Next is model evaluation. After performing all these we got test accuracy to be 79% and validation accuracy as 81%.

4. IMPLEMENTATION

4.1 Neural network coding

4.1.1 Importing essential packages

```
import numpy as  np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import os from glob

import glob

from PIL import Image from sklearn.preprocessing

import label_binarize

from sklearn.metrics import confusion_matrix

import itertools

import keras

from keras.utils.np_utils import to_categorical

from keras.models import Sequential

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

from keras import backend as K

import itertools

from keras.layers.normalization import BatchNormalization

from keras.utils.np_utils import to_categorical

from keras.optimizers import Adam

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import ReduceLROnPlateau

from sklearn.model_selection import train_test_split
```

4.1.2 Loading dataset

```
skin_df=    pd.read_csv("/content/drive/MyDrive/skinLesionDataset/HAM10000_metadata.csv")

base_path = "/content/drive/MyDrive/skinLesionDataset/images"

imageid_path_dict = {}
```

```
for x in glob(os.path.join(base_path, '*', '*.jpg')):
    imageid_path_dict[os.path.splitext(os.path.basename(x))[0]] = x
```

4.1.3 Reducing memory usage

```
def reduce_mem_usage(df, verbose=True):

    numerics = ['int16', 'int32', 'int64', 'float16',
    'float32'] start_mem = df.memory_usage().sum() /
    1024**2 for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max <
                    np.finfo(np.float16).max: df[col] = df[col].astype(np.float16)
                elif c_min> np.finfo(np.float32).min and c_max <
                    np.finfo(np.float32).max: df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage().sum() / 1024**2
```

print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))

print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

return df

```
skin_df = pd.read_csv("/content/drive/MyDrive/skinLesionDataset/HAM10000_metadata.csv")
skin_df = reduce_mem_usage(skin_df)

Memory usage after optimization is: 0.48 MB
Decreased by 10.7%
```

Fig4.1.3 Memory reduce

4.1.4 Expanding dataset

lesion_type_dict = {

'nv': 'Melanocytic nevi',

'mel': 'Melanoma',

'bkl': 'Benign keratosis-like lesions ',

'bcc': 'Basal cell carcinoma',

'akiec': 'Actinic keratoses',

'vasc': 'Vascular lesions',

'df': 'Dermatofibroma'

}

skin_df['path'] = skin_df['image_id'].map(imageid_path_dict.get)

skin_df['cell_type'] = skin_df['dx'].map(lesion_type_dict.get)

skin_df['cell_type_idx'] = pd.Categorical(skin_df['cell_type']).codes

```
skin_df.sample(5)
```

	lesion_id	image_id	dx	dx_type	age	sex	localization	path	cell_type	cell_type_idx
3602	HAM_0005241	ISIC_0029699	nv	follow_up	30.0	male	trunk	None	Melanocytic nevi	4
6461	HAM_0004686	ISIC_0028574	nv	follow_up	40.0	female	abdomen	None	Melanocytic nevi	4
1739	HAM_0006824	ISIC_0033538	mel	histo	70.0	male	lower extremity	None	Melanoma	5
1019	HAM_0005993	ISIC_0026441	bkl	consensus	80.0	male	lower extremity	None	Benign keratosis-like lesions	2
8094	HAM_0001012	ISIC_0031867	nv	histo	55.0	male	chest	None	Melanocytic nevi	4

Fig.4.1.4 Extended dataset

4.1.5 Data preprocessing

Data Cleaning

skin_df.isnull().sum()

skin_df['age'].fillna((skin_df['age'].mean()), inplace=True)

Resizing images according to tensorflow passable size

 skin_df['image] = skin_df['path'].map(lambda x:
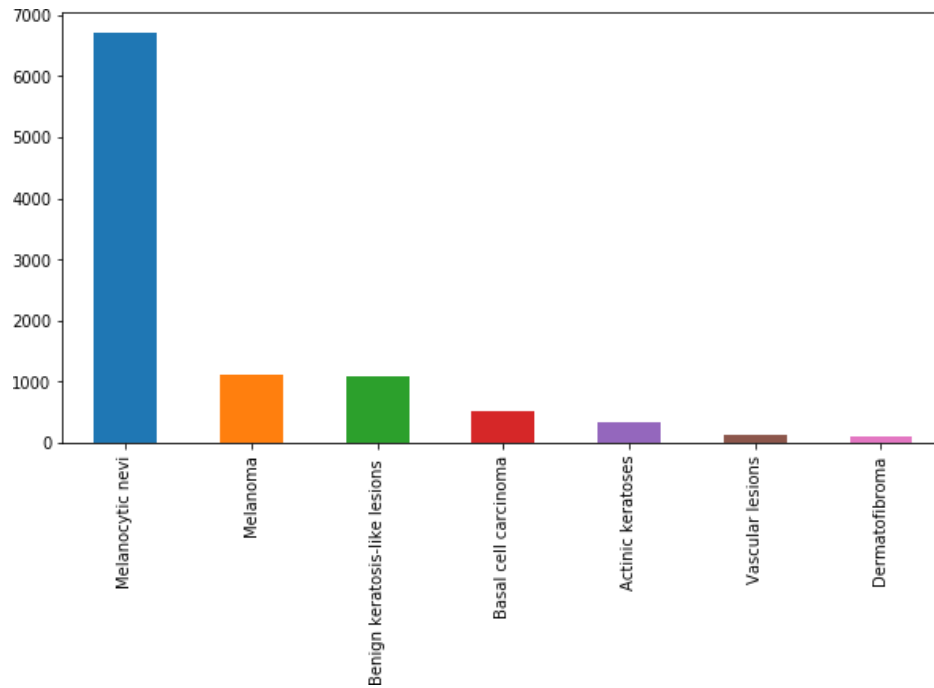
np.asarray(Image.open(x).resize((224, 224))))

4.1.6 EDA

#Comparing number of cases of each lesion fig,

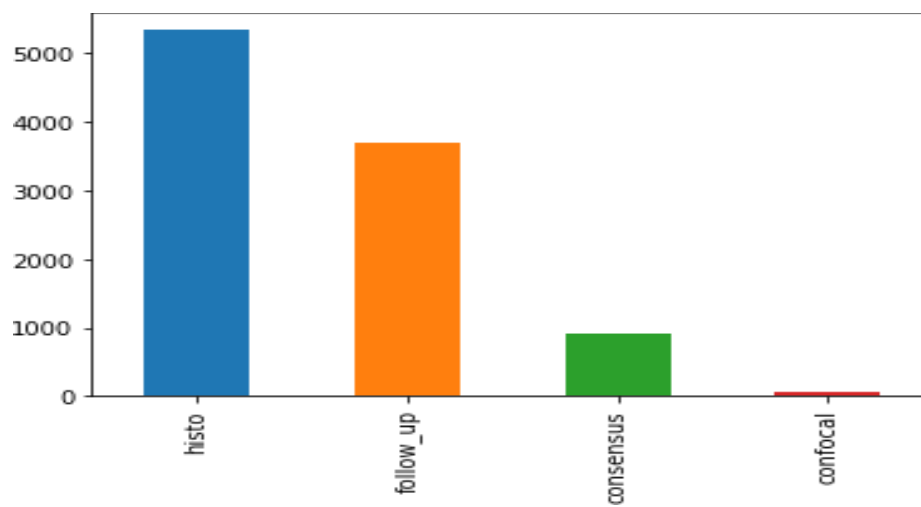ax1 = plt.subplots(1, 1, figsize= (10, 5))

skin_df['cell_type'].value_counts().plot(kind='bar', ax=ax1)

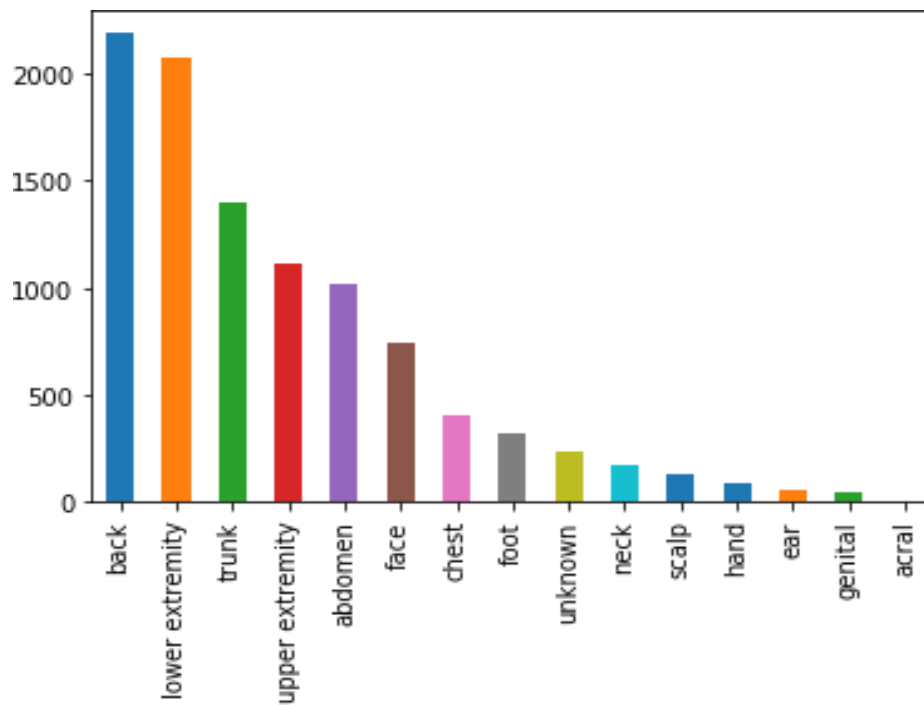**Fig.4.1.6.1** Bar Graph of comparing Lesion

#Comparing tests counts

skin_df['dx_type'].value_counts().plot(kind='bar')

Fig. 4.1.6.2 Bar Graph of comparing tests

#Comparing localization counts

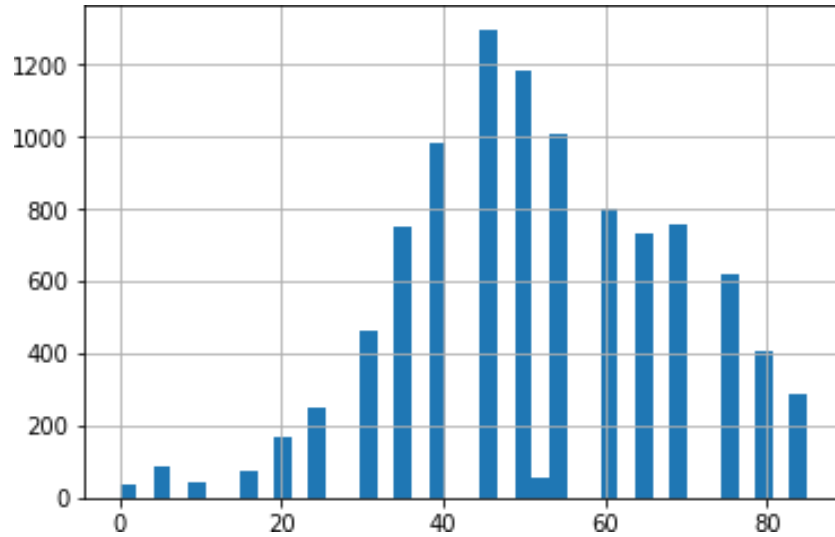skin_df['localization].value_counts().plot(kind='bar')

Fig. 4.1.6.3 Bar graph for localization

#Plotting age vs count of people with lesions
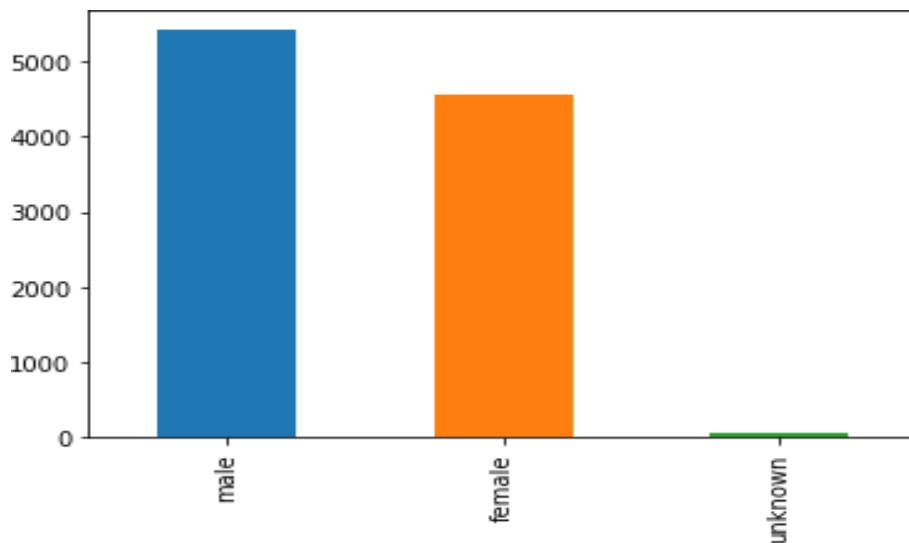
skin_df['age'].hist(bins = 40)

Fig.4.1.6.4 Histogram for Plotting age Vs Count of people with Lesion

Comparing gender vs people with lesions

skin_df['sex'].value_counts().plot(kind='bar')

Fig. 4.1.6.5 Bar Graph for comparing Gender Vs people with Lesion

\# Age vs lesion type

sns.scatterplot('age', 'cell_type_idx', data = skin_df)
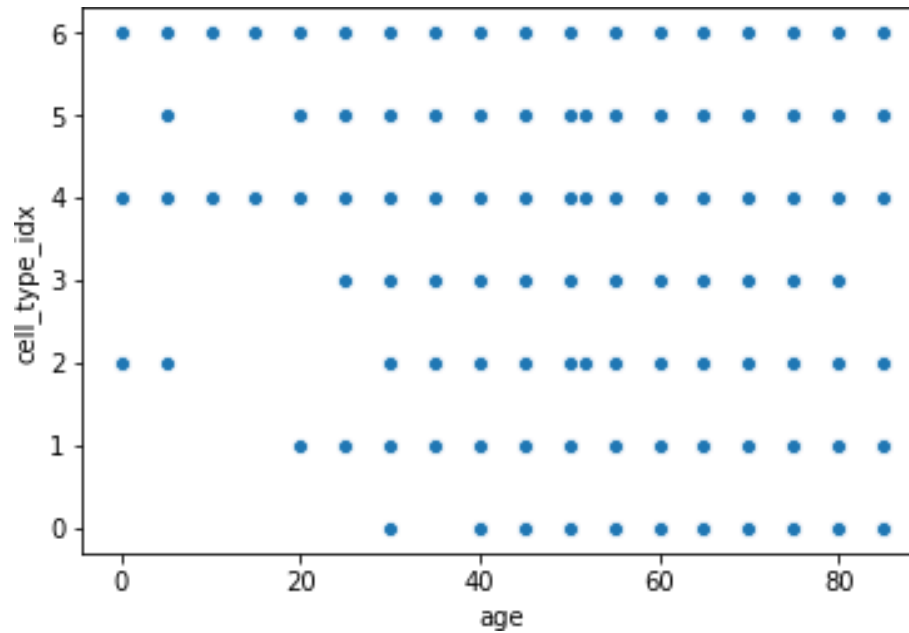
Fig. 4.1.6.6 Scatterplot Age Vs Lesion type

4.1.7 Train – Test data split

features = skin_df.drop(columns=['cell_type_idx'], axis=1)

target = skin_df['cell_type_idx']

x_train_o, x_test_o, y_train_o, y_test_o = train_test_split(features, target, test_size=0.20, random_state=1234)

4.1.8 Normalization

x_train = np.asarray(x_train_o['image'].tolist())

x_test = np.asarray(x_test_o['image'].tolist())

 x_train_mean = np.mean(x_train)

x_train_std = np.std(x_train)

x_test_mean = np.mean(x_test)

```
x_test_std = np.std(x_test)

x_train = (x_train - x_train_mean)/x_train_std

x_test = (x_test - x_test_mean)/x_test_std

y_train = to_categorical(y_train_o, num_classes=7)

y_test = to_categorical(y_test_o, num_classes=7)
```

4.1.9 Mobilenet CNN model building

```
K.image_data_format()

def mobilenet_block(x, filters, strides):

x = DepthwiseConv2D(kernel_size=3, strides=strides, padding='same')(x) x =

BatchNormalization()(x)

x = ReLU()(x)

x = Conv2D(filters=filters, kernel_size=1, strides=1, padding='same')(x)

x = BatchNormalization()(x)

x = ReLU()(x)

return x


INPUT_SHAPE = 224, 224, 3

input = Input(INPUT_SHAPE)

x = Conv2D(filters=32, kernel_size=3, strides=2, padding='same')(input) x =

BatchNormalization()(x)

x = ReLU()(x)

x = mobilenet_block(x, filters=64, strides=1)

x = mobilenet_block(x, filters=128, strides=2)

x = mobilenet_block(x, filters=128, strides=1)

x = mobilenet_block(x, filters=256, strides=2)

x = mobilenet_block(x, filters=256, strides=1)

x = mobilenet_block(x, filters=512,  strides=2)
```

for _ in range(5):

x = mobilenet_block(x, filters=512, strides=1)

x = mobilenet_block(x, filters=1024, strides=2)

x = mobilenet_block(x, filters=1024, strides=1)

x = AvgPool2D(pool_size=7, strides=1)(x) flattened = Flatten()(x)

output = Dense(units=7, activation='softmax')(flattened)

model = Model(inputs=input, outputs=output) model.summary()

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fig.4.1.9 MobileNet model

```
#Optimizer
optimizer = Adam(lr=0.001,
beta_1=0.9, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=False)



#Compiling the model
model.compile(optimizer = optimizer,
loss = "categorical_crossentropy", metrics = ["accuracy"])



#Setting a learning rate annealer
leraning_rate_reduction = ReduceLROnPlateau(monitor='accuracy' , patience=3, verbose=1,
factor=0.5, min_lr=0.00001)



#Data augmentation for preventing overfitting
datagen = ImageDataGenerator( featurewise_center=False, samplewise_center=False,
featurewise_std_normalization=False, samplewise_std_normalization=False,
zca_whitening=False, rotation_range=10, zoom_range = 0.1, width_shift_range=0.1,
height_shift_range=0.1,  horizontal_flip=False,  vertical_flip=False)
datagen.fit(x_train)
```

4.1.10 Training the model

```
epochs = 15
batch_size = 20
history = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
epochs = epochs, validation_data = (x_validate,y_validate), verbose = 1,
steps_per_epoch=x_train.shape[0] // batch_size , callbacks=[learning_rate_reduction])
```

4.1.11 Calculating accuracy

loss, accuracy = model.evaluate(x_test, y_test, verbose=1)

loss_v, accuracy_v = model.evaluate(x_validate, y_validate, verbose=1)

print("Validation: accuracy = %f; loss_v = %f" %(accuracy_v, loss_v)

print("Test: accuracy = %f; loss_v = %f" %(accuracy, loss)

```
loss, accuracy = model.evaluate(x_test, y_test, verbose=1)
loss_v, accuracy_v = model.evaluate(x_validate, y_validate, verbose=1)

print("Validation: accuracy = %f  ;  loss_v = %f" % (0.813117, 0.722661))
print("Test: accuracy = %f  ;  loss = %f" % (0.798877, 0.684366))

Validation: accuracy = 0.813117  ;  loss_v = 0.722661
Test: accuracy = 0.798877  ;  loss = 0.684366
```

Fig.4.1.11 Calculating Accuracy

4.1.12 Plotting confusion matrix

def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix',

cmap=plt.cm.Blues):

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick_marks = np.arange(len(classes))

plt.xticks(tick_marks, classes, rotation=45)

plt.yticks(tick_marks, classes)

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:,

np.newaxis] thresh = cm.max() / 2

tick_marks = np.arange(len(classes))

plt.xticks(tick_marks, classes, rotation=45)

plt.yticks(tick_marks, classes)

```
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                plt.text(j, i, cm[i, j], horizontalalignment="center", color="white"
                if cm[i, j] > thresh else "black")
        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')


# Predict the values from the validation dataset
Y_pred = model.predict(x_validate)
# Convert predictions classes to one hot
vectors Y_pred_classes=
np.argmax(Y_pred,axis = 1)
# Predict the values from the validation
dataset
Y_pred = model.predict(x_validate)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_validate,axis = 1)
# compute the confusion matrix

confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)

# Convert predictions classes to one hot
vectors Y_pred_classes =
np.argmax(Y_pred,axis = 1)

# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes =range(7))


# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes =range(7))
```
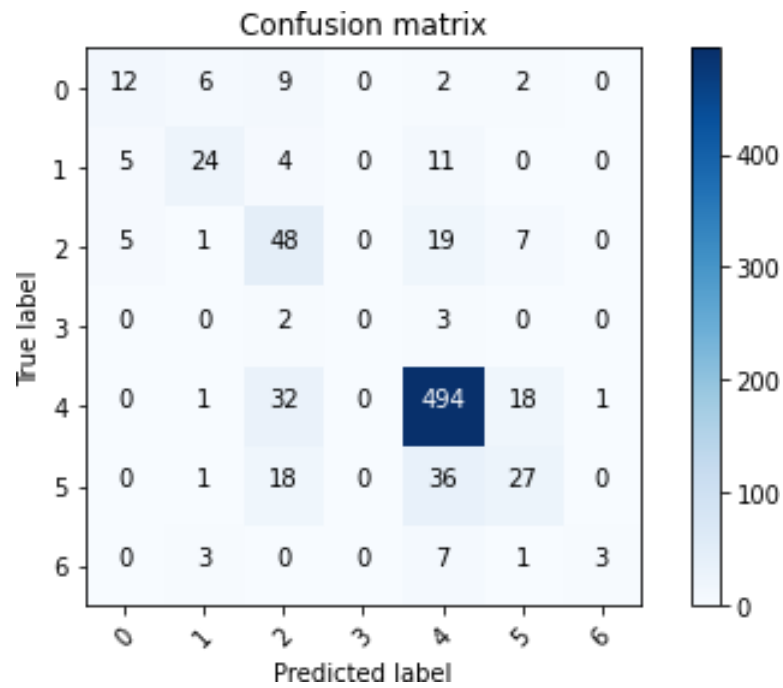
Fig.4.1.12 Confusion Matrix

4.2 UI(User Interface) CODING

4.2.1 Importing essential packages

```
import tkinter as tk
from PIL import ImageTk, Image
from tkinter import filedialog
from tkinter import Label
from keras.models import load_model
from matplotlib import pyplot as plt
import cv2
import numpy as np
from tkinter.messagebox import *
from tkinter import *
```

4.2.2 Prediction code

```
model=load_model("/home/adminpc/Downloads/skinLesionmodel.h5")
def UploadAction(event=None):
        filename = filedialog.askopenfilename()
        #print(filename)


        img = cv2.imread(filename)
        load = Image.open(filename)
        load = load.resize((300, 300), Image.ANTIALIAS)
        render = ImageTk.PhotoImage(load)
        img2 = Label(image=render)
        img2.image = render
        img2.place(x=100, y=160)


        img.resize(1, 75, 100, 3)

        les_type = np.argmax(model.predict(img), axis=-1)
        #print(les_type)
```

```
Label(root, text="The lesion type is: ").place(x=100, y = 500)

Label(root, text=lesionTypeMap[les_type[0]]).place(x=300,
y=500)
```

4.2.3 User Interface

```
root = tk.Tk()

root.title("Skin Lesion Classification")

root.minsize(750, 1000)

root.resizable(width=True, height = True)


req_font = ("MS Serif", 12)

heading = Label(root, text="Select image").place(x=100, y=100)

heading.configure(font=req_font)

button = tk.Button(root, text='Upload image',

command=UploadAction).place(x=100, y=125)

root.mainloop()
```

4.3 TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and coding. The increasing visibility of software as a system element and attendant costs associated with a software failure are motivating factors for we planned, through testing.

4.3.1 Test Cases

Test Case ID	Test Scenario	Expected Result	Actual Result	Pass/Fail
TC01	Check if User window is opened	User window must be opened	As expected	Pass
TC02	Check if upload image button working	Image should be taken from the selected folder	As expected	Pass
TC03	Check if image is being displayed or not	Image should be displayed	As expected	Pass
TC04	Check if Image is detected or not	A statement must be displayed(Type of lesion)	As expected	Pass
TC05	Check if Lesion type is displayed	Detected Lesion type to be displayed	As expected	Pass

Fig.4.3.1 Test cases

4.4 DATASET TRAINING IMAGES

After tuning various parameters and compiled the model we fit the model using 15 epochs with a batch size of 20.

The number of epochs is a hyperparameter that defines the number of times that the learning algorithm will work through the entire training dataset. Batch size is the number of samples processed before the model is updated. The number of epochs is the number of complete passes through the training dataset.

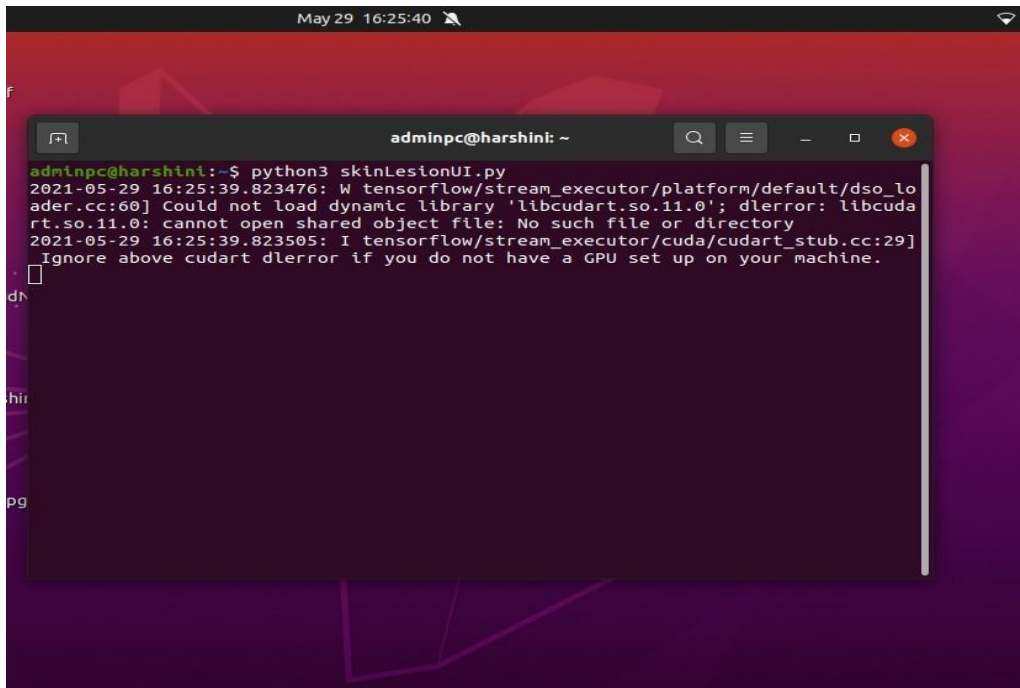So we chose the epochs and batch size in such a way that our model could train sufficiently.

```
Epoch 1/15
721/721 [==============================] - 975s 1s/step - loss: 0.7816 - accuracy: 0.7103 - val_loss: 1.8912 - val_accuracy: 0.5449
Epoch 2/15
721/721 [==============================] - 1001s 1s/step - loss: 0.7752 - accuracy: 0.7155 - val_loss: 2.9008 - val_accuracy: 0.3741
Epoch 3/15
721/721 [==============================] - 1016s 1s/step - loss: 0.7594 - accuracy: 0.7190 - val_loss: 1.0029 - val_accuracy: 0.7070
Epoch 4/15
721/721 [==============================] - 1018s 1s/step - loss: 0.7454 - accuracy: 0.7237 - val_loss: 0.7745 - val_accuracy: 0.7344
Epoch 5/15
721/721 [==============================] - 1000s 1s/step - loss: 0.7481 - accuracy: 0.7280 - val_loss: 1.1367 - val_accuracy: 0.6983
Epoch 6/15
721/721 [==============================] - 998s 1s/step - loss: 0.7208 - accuracy: 0.7351 - val_loss: 1.0480 - val_accuracy: 0.6758
Epoch 7/15
721/721 [==============================] - 994s 1s/step - loss: 0.7134 - accuracy: 0.7369 - val_loss: 0.7240 - val_accuracy: 0.7394
Epoch 8/15
721/721 [==============================] - 982s 1s/step - loss: 0.6972 - accuracy: 0.7413 - val_loss: 1.0357 - val_accuracy: 0.6958
Epoch 9/15
721/721 [==============================] - 985s 1s/step - loss: 0.6867 - accuracy: 0.7463 - val_loss: 0.9302 - val_accuracy: 0.7207
Epoch 10/15
721/721 [==============================] - 983s 1s/step - loss: 0.6919 - accuracy: 0.7447 - val_loss: 0.7220 - val_accuracy: 0.7332
Epoch 11/15
721/721 [==============================] - 991s 1s/step - loss: 0.6734 - accuracy: 0.7534 - val_loss: 0.7198 - val_accuracy: 0.7394
Epoch 12/15
721/721 [==============================] - 995s 1s/step - loss: 0.6672 - accuracy: 0.7577 - val_loss: 0.7418 - val_accuracy: 0.7282
Epoch 13/15
721/721 [==============================] - 995s 1s/step - loss: 0.6522 - accuracy: 0.7638 - val_loss: 0.7631 - val_accuracy: 0.7182
Epoch 14/15
721/721 [==============================] - 887s 1s/step - loss: 0.6363 - accuracy: 0.7674 - val_loss: 0.8342 - val_accuracy: 0.7244
Epoch 15/15
721/721 [==============================] - 884s 1s/step - loss: 0.6208 - accuracy: 0.7691 - val_loss: 0.7227 - val_accuracy: 0.7531
```

Fig.4.4.1 Dataset Training Images

4.5 INPUT IMAGES

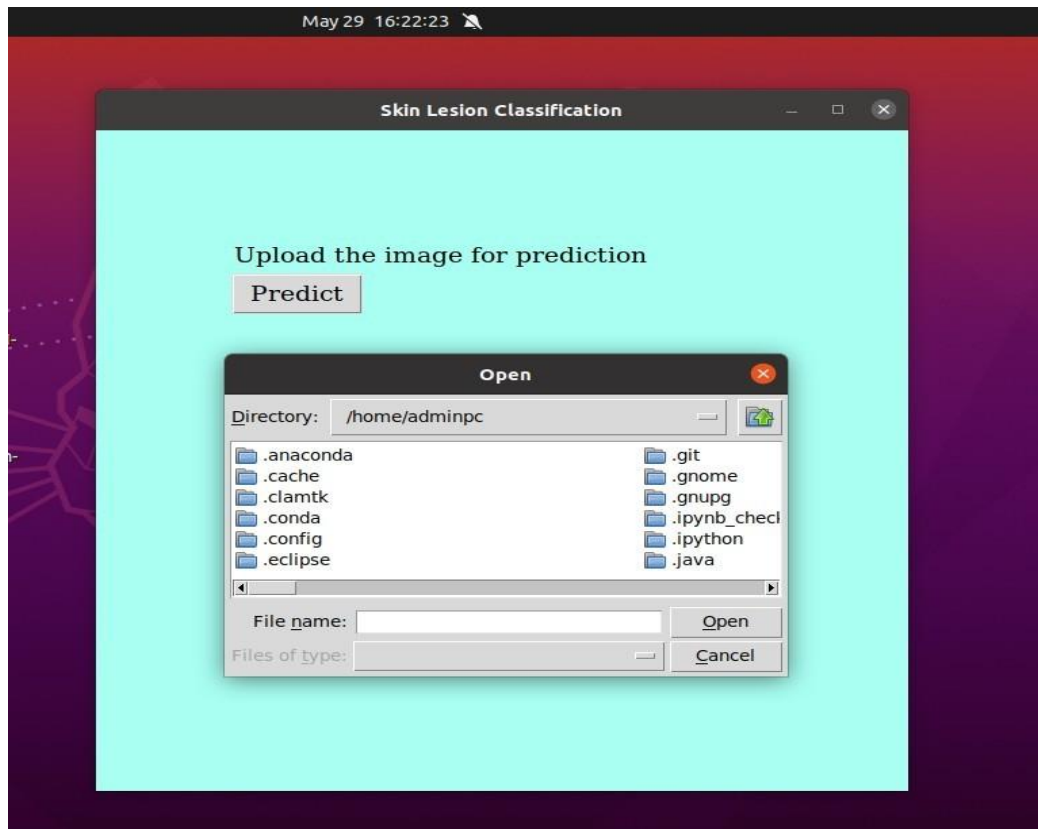Command to run the project

Fig.4.5.1 Command to run the Project
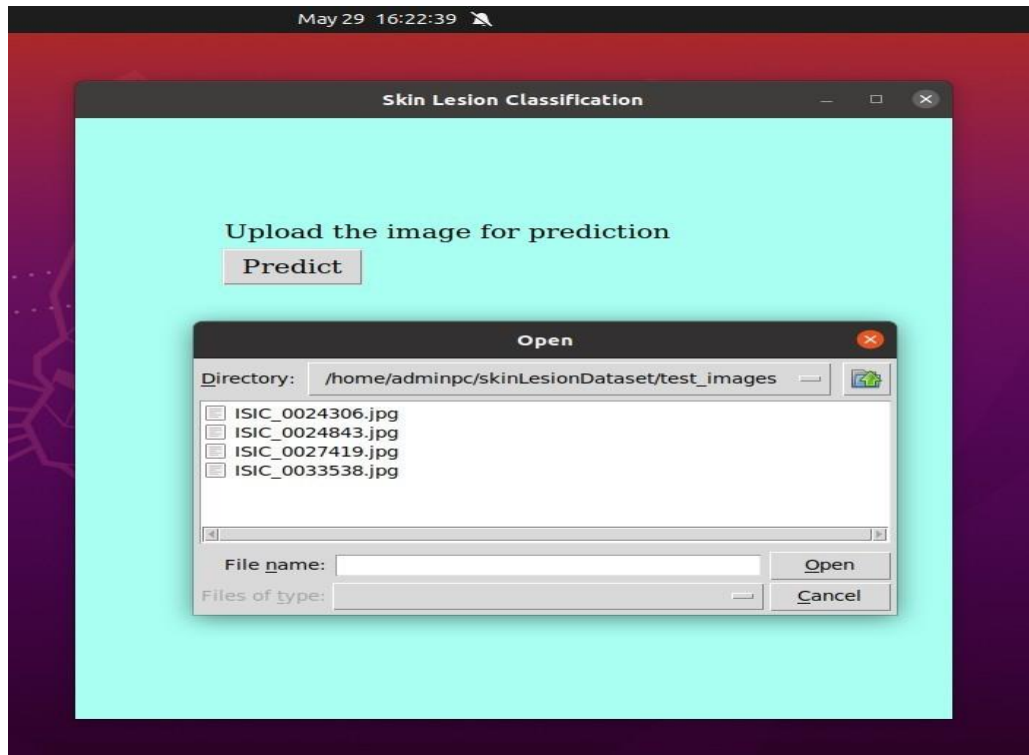
Upload the images

Fig.4.5.2 Upload the images
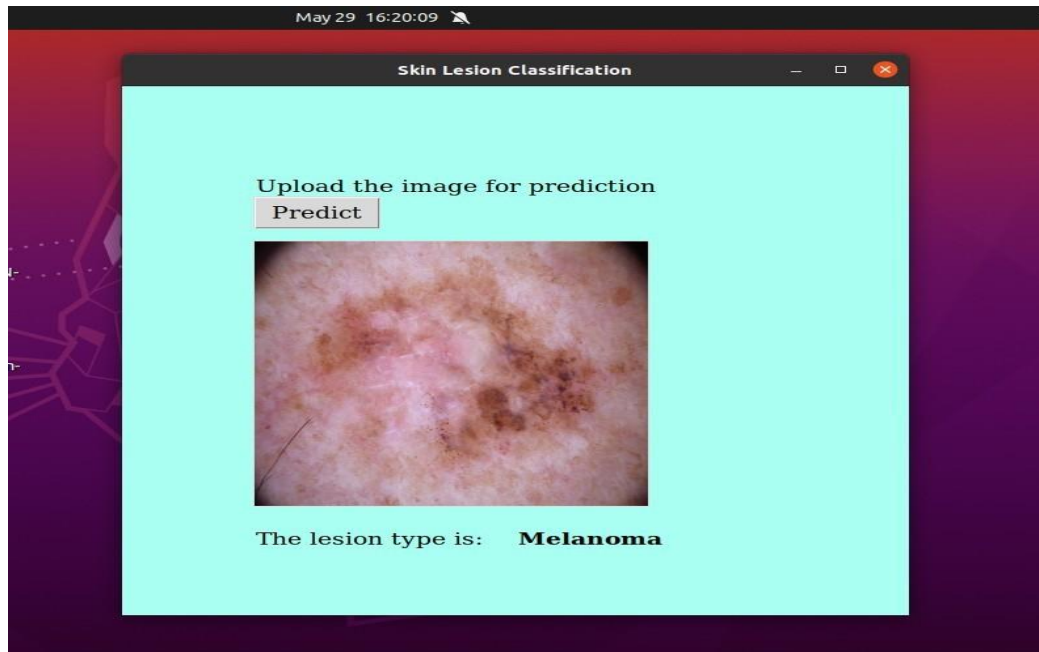
Select the file with images

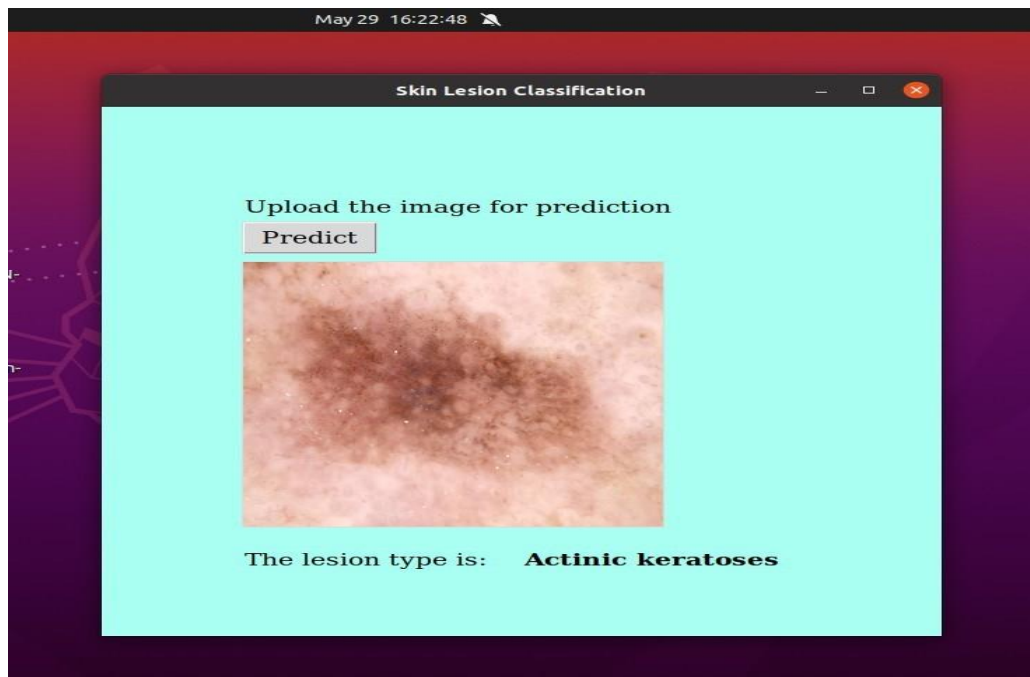**Fig.4.5.3** Selecting file to upload image

Select the image

Fig.4.5.4 Selecting Image to upload

4.6 OUTPUT IMAGES

Fig.4.6.1 Predicting Lesion Type(1)

Fig.4.6.2 Predicting Lesion Type(2)

5. CONCLUSION AND FUTURE SCOPE

This application helps a doctor to classify skin lesions to their respective type among 7 classes accurately. Skin cancer is a major threat to human health and early identification can save a person's life but this rapid classification/identification remains difficult in many parts of the world due to the lack of necessary infrastructure. Training of neural networks for automated diagnosis of pigmented skin lesions is hampered by the small size and lack of diversity of available datasets of dermatoscopic images. This problem was tackled by the HAM10000 dataset which was formed by collecting images from different populations, acquired and stored by different modalities. After the training of the model using a deep convolutional neural network, we achieved an accuracy of 81%.

The future enhancement of this application could be to further increase the accuracy present a treatment plan.

6. REFERENCES

[1] Saket S. Chaturvedi, Kajol Gupta, Prakash. S. Prasad, "Skin Lesion Analyser: An Efficient Seven-Way Multi-Class Skin Cancer Classification Using MobileNet", May. 2020. (BasePaper) https://arxiv.org/abs/1907.03220

[2] M. A. A. Milton, "Automated Skin Lesion Classification Using Ensemble of Deep Neural Networks in ISIC 2018: Skin Lesion Analysis Towards Melanoma Detection Challenge," Jan. 2019.

[3] Kawahara J, Hamarneh G. Multi-resolution-Tract CNN with Hybrid Pretrained and Skin Lesion Trained Layers. In: Wang L, Adeli E, Wang Q, Shi Y, Suk HI (eds) Machine Learning in Medical Imaging. MLMI 2016. https://doi.org/10.1007/978-3-319-47157-0_20

[4] Burroni M et al. Melanoma computer-aided diagnosis: reliability and feasibility study. Clin. Cancer Res. 2004; https://doi.org/10.1158/1078-0432.CCR-03-0039

Kawahara J, Hamarneh G. Multi-resolution-Tract CNN with Hybrid Pretrained and SkinLesion Trained Layers. In: Wang L, Adeli E, Wang Q, Shi Y, Suk HI Machine Learning. https://doi.org/10.1007/978-3-319-47157-0_20

[5] M. A. A. Milton, "Automated Skin Lesion Classification Using Ensemble of Deep Neural Networks in ISIC 2018: Skin Lesion Analysis Towards Melanoma Detection Challenge".

Howard AG et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv Prepr. https://arxiv.org/abs/1704.04861. 2017.

[6] "A Mobile Automated Skin Lesion Classification System" by Kiran Ramlakhan, Yi Shang in 2011 IEEE 23rd International Conference. https://ieeexplore.ieee.org/abstract/document/6103318

[7] Adria Romero Lopez, Xavier Giro-i-Nieto, Jack Burdick, Oge Marques "Skin lesion classification from dermoscopic images using deep learning techniques" in 2017 13th International Conference on Biomedical Engineering, IEEE. https://ieeexplore.ieee.org/xpl/conhome/7890909/proceeding

[8] Jordan Yap, William Yolland, Philipp Tschandl "Multimodal skin lesion classification using deep learning" https://doi.org/10.1111/exd.13777