Dermatologist-Level Classification of Skin Cancer Using Cascaded Ensembling of Convolutional Neural Network and Handcrafted Features Based Deep Neural Network

ABSTRACT

ABSTRACT

Skin cancer is caused due to unusual development of skin cells and deadly type cancer. Early diagnosis is very significant and can avoid some categories of skin cancers, such as melanoma and focal cell carcinoma. The recognition and the classification of skin malignant growth in the beginning time is expensive and challenging. The deep learning architectures such as recurrent networks and convolutional neural networks (ConvNets) are developed in the past, which are proven appropriate for non-handcrafted extraction of complex features. To additional expand the efficiency of the ConvNet models, a cascaded ensembled network that uses an integration of ConvNet and handcrafted features based multi-layer perceptron is proposed in this work. This offered model utilizes the convolutional neural network model to mine non-handcrafted image features and colour moments and texture features as handcrafted features. It is demonstrated that accuracy of ensembled deep learning model is improved to 98:3% from 85:3% of convolutional neural network model.

TABLE OF CONTENTS

LIST OF FIGURE

INTRODUCTION

1.INTRODUCTION

The skin is the major tissue of the human body that covers about twenty square feet area. It covers the complete body and its thickness differs significantly over all parts of the body, and also varies between man and woman and the old and young one. For example, the average thickness of the skin on the forearm is 1.26 mm in female and 1.3 mm in male. The skin shields against thermal, mechanical, and bodily harms. It also defends us against bacteria and the elements, and the presence of intercellular lipids prevents moisture loss. Over the last few decades, there has been an upsurge in the number of patients diagnosed with skin cancer.. Skin cancer sufferers must have early detection and regular diagnosis in order to survive. Though, a significant number of cases remain unobserved until it reach to advanced stages, which reduces the chances of survival. An appealing method for early recognition is to employ automated classification of dermoscopic images analysed via Computer Based Diagnosis (CBD) system, [6]. CBD is basically clinical decision support system that assists clinicians in the understanding of medical images. CBD is used as an instrument to deliver additional information to dermatologist, who takes final decision. Its primary goal is to increase the diagnosis accuracy and consistency of dermatologist by decreasing the false negative rate due to observational oversight, intra-observer and inter-observer variation. Most of the time two types of broad methodologies are deployed in CBD systems. The first stage is to get the position of the lesions. The next stage is to quantify the image features of abnormal and/or normal patterns. Usually, the computer based diagnosis system includes three basic components. The foremost is the image processing and analysis system that supports to enhance and extract the lesions by selection of the primary candidates of the lesions and apprehensive patterns. The second is the quantification of image features for example the size, colour, texture, shape and contrast of the pigments selected in the first step. It is essential to identify distinctive features that can discriminate consistently between a lesion and other usual anatomical structures. The last stage is feature processing which classifies between abnormal and normal patterns or identify skin lesion class, based on the features acquired in the second stage.

1.1 SOFTWARE REQUIREMENTS

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Platform – In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming languages and their runtime libraries.

Operating system is one of the first requirements mentioned when defining system requirements (software). Software may not be compatible with different versions of same line of operating systems, although some measure of backward compatibility is often maintained. For example, most software designed for Microsoft Windows XP does not run on Microsoft Windows 98, although the converse is not always true. Similarly, software designed using newer features of Linux Kernel v2.6

generally does not run or compile properly (or at all) on Linux distributions using Kernel v2.2 or v2.4.

APIs and drivers – Software making extensive use of special hardware devices, like high-end display adapters, needs special API or newer device drivers. A good example is DirectX, which is a collection of APIs for handling tasks related to multimedia, especially game programming, on Microsoft platforms.

Web browser – Most web applications and software depending heavily on Internet technologies make use of the default browser installed on system. Microsoft Internet Explorer is a frequent choice of software running on Microsoft Windows, which makes use of ActiveX controls, despite their vulnerabilities.

1)Visual Studio Community Version

2)Nodejs (Version 12.3.1)

3)Python IDEL (Python 3.7)

1.2 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

Architecture – All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture. See also a list of common operating systems and their supporting architectures.

Processing power – The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored. This definition of power is often erroneous, as AMD Athlon and Intel Pentium CPUs at similar clock speed often have different throughput speeds. Intel Pentium CPUs have enjoyed a considerable degree of popularity, and are often mentioned in this category.

Memory – All software, when run, resides in the random access memory (RAM) of a computer. Memory requirements are defined after considering demands of the application, operating system, supporting software and files, and other running processes. Optimal performance of other unrelated software running on a multi-tasking computer system is also considered when defining this requirement.

Secondary storage – Hard-disk requirements vary, depending on the size of software installation, temporary files created and maintained while installing or running the software, and possible use of swap space (if RAM is insufficient).

Display adapter – Software requiring a better than average computer graphics display, like graphics editors and high-end games, often define high-end display adapters in the system requirements.

Peripherals – Some software applications need to make extensive and/or special use of some peripherals, demanding the higher performance or functionality of such peripherals. Such peripherals include CD-ROM drives, keyboards, pointing devices, network devices, etc.

1)Operating System : Windows Only

2)Processor : i5 and above

3)Ram : 4gb and above

4)Hard Disk : 50 GB

FEASIBILITY STUDY

2. FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ♦ ECONOMICAL FEASIBILITY
- ♦ TECHNICAL FEASIBILITY
- ♦ SOCIAL FEASIBILITY

2.1 ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

2.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.3 SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

LITERATURE SURVEY

3.LITERATURE SURVEY

3.1 Non melanoma skin cancer pathogenesis overview:

https://www.mdpi.com/2227-9059/6/1/6

ABSTRACT: Non-melanoma skin cancer is the most frequently diagnosed cancer in humans. The process of skin carcinogenesis is still not fully understood. However, several studies have been conducted to better explain the mechanisms that lead to malignancy; (2) Methods: We reviewed the more recent literature about the pathogenesis of non-melanoma skin cancer focusing on basal cell carcinomas, squamous cell carcinoma and actinic keratosis; (3) Results: Several papers reported genetic and molecular alterations leading to non-melanoma skin cancer. Plenty of risk factors are involved in non-melanoma skin cancer pathogenesis, including genetic and molecular alterations, immunosuppression, and ultraviolet radiation; (4) Conclusion: Although skin carcinogenesis is still not fully understood, several papers demonstrated that genetic and molecular alterations are involved in this process. In addition, plenty of non-melanoma skin cancer risk factors are now known, allowing for an effective prevention of non-melanoma skin cancer development. Compared to other papers on the same topic, our review focused on molecular and genetic factors and analyzed in detail several factors involved in non-melanoma skin cancer.

3.2 Skin cancer classification using convolutional neural networks: Systematic review

https://www.researchgate.net/publication/327539277_Skin_Cancer_Classification_using_Convolutional_Neural_Networks_Systematic_Review_Preprint

ABSTRACT: State-of-the-art classifiers based on convolutional neural networks (CNNs) were shown to classify images of skin cancer on par with dermatologists and could enable lifesaving and fast diagnoses, even outside the hospital via installation of apps on mobile devices. To our knowledge, at present there is no review of the current work in this research area. Objective: This study presents the first systematic review of the state-of-the-art research on classifying skin lesions with CNNs. We limit our review to skin lesion classifiers. In particular, methods that apply a CNN only for segmentation or for the classification of dermoscopic patterns are not considered here. Furthermore, this study discusses why the comparability of the presented procedures is very difficult and which challenges must be addressed in the future. Methods: We searched the Google Scholar, PubMed, Medline, ScienceDirect, and Web of Science databases for systematic reviews and original research articles published in English. Only papers that reported sufficient scientific proceedings are included in this review. Results: We found 13 papers that classified skin lesions using CNNs. In principle, classification methods can be differentiated according to three principles. Approaches that use a CNN already trained by means of another large dataset and then optimize its parameters to the classification of skin lesions are the most common ones used and they display the best performance with the currently available limited datasets. Conclusions: CNNs display a high performance as state-of-the-art skin lesion classifiers. Unfortunately, it is difficult to compare different classification methods because some approaches use nonpublic datasets for training and/or testing, thereby making reproducibility difficult. Future publications should use publicly available benchmarks and fully disclose methods used for training to allow comparability.

3.3 Human–computer collaboration for skin cancer recognition

https://www.nature.com/articles/s41591-020-0942-0

ABSTRACT: The rapid increase in telemedicine coupled with recent advances in diagnostic artificial intelligence (AI) create the imperative to consider the opportunities and risks of inserting AI-based support into new paradigms of care. Here we build on recent achievements in the accuracy of image-based AI for skin cancer diagnosis to address the effects of varied representations of AI-based support across different levels of clinical expertise and multiple clinical workflows. We find that good quality AI-based support of clinical decision-making improves diagnostic accuracy over that of either AI or physicians alone, and that the least experienced clinicians gain the most from AI-based support. We further find that AI-based multiclass probabilities outperformed content-based image retrieval (CBIR) representations of AI in the mobile technology environment, and AI-based support had utility in simulations of second opinions and of telemedicine triage. In addition to demonstrating the potential benefits associated with good quality AI in the hands of non-expert clinicians, we find that faulty AI can mislead the entire spectrum of clinicians, including experts. Lastly, we show that insights derived from AI class-activation maps can inform improvements in human diagnosis. Together, our approach and findings offer a framework for future studies across the spectrum of image-based diagnostics to improve human–computer collaboration in clinical practice.

3.4 Management of primary skin cancer during a pandemic: Multidisciplinary recommendations

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7301000/

ABSTRACT: During the coronavirus disease 2019 (COVID‐19) pandemic, providers and patients must engage in shared decision making regarding the pros and cons of early versus delayed interventions for localized skin cancer. Patients at highest risk of COVID‐19 complications are older; are immunosuppressed; and have diabetes, cancer, or cardiopulmonary disease, with multiple comorbidities associated with worse outcomes. Physicians must weigh the patient's risk of COVID‐19 complications in the event of exposure against the risk of worse oncologic outcomes from delaying cancer therapy. Herein, the authors have summarized current data regarding the risk of COVID‐19 complications and mortality based on age and comorbidities and have reviewed the literature assessing how treatment delays affect oncologic outcomes. They also have provided multidisciplinary recommendations regarding the timing of local therapy for early‐stage skin cancers during this pandemic with input from experts at 11 different institutions. For patients with Merkel cell carcinoma, the authors recommend prioritizing treatment, but a short delay can be considered for patients with favorable T1 disease who are at higher risk of COVID‐19 complications. For patients with melanoma, the authors recommend delaying the treatment of patients with T0 to T1 disease for 3 months if there is no macroscopic residual disease at

the time of biopsy. Treatment of tumors ≥T2 can be delayed for 3 months if the biopsy margins are negative. For patients with cutaneous squamous cell carcinoma, those with Brigham and Women's Hospital T1 to T2a disease can have their treatment delayed for 2 to 3 months unless there is rapid growth, symptomatic lesions, or the patient is immunocompromised. The treatment of tumors ≥T2b should be prioritized, but a 1‐month to 2‐month delay is unlikely to worsen disease‐specific mortality. For patients with squamous cell carcinoma in situ and basal cell carcinoma, treatment can be deferred for 3 months unless the individual is highly symptomatic.

3.5 Development of mobile skin cancer detection using faster R-CNN and MobileNet v2 model

https://ieeexplore.ieee.org/document/9239197

ABSTRACT: The development of cameras in smartphones is possible as a point of care for early detection of cancer. Early detection using smartphones is carried out by giving the smartphone the ability to recognize objects with skin cancer characteristics. The convolution neural network (CNN) is often used in disease detection and classification. However, the CNN method requires high computing capability and a large memory that is difficult to perform on smartphones. In this paper, the MobileNet v2 and Faster R-CNN methods are utilized and executed on an Android-based application that can detect skin cancer. Both proposed architectures were trained to recognize actinic keratosis and melanoma skin cancer targets on images. The dataset used was 600 images, divided into two classes, actinic keratosis images and melanoma images with no attention for gender, age, or additional factors. In this study, an android app was developed to utilize the smartphone camera for skin cancer detection. The Faster R-CNN and MobileNet v2 models were implemented as an intelligent system for the screening. Two testing methods were performed in this study, the Jupyter notebook and android camera. Based on the experiment result, Faster R-CNN obtained higher accuracy when testing using the Jupyter, and MobileNet v2 got the same high accuracy when applying on a smartphone.

SYSTEM ANALYSIS

4.SYSTEM ANALYSIS

4.1 EXISTING SYSTEM:

Skin cancer is caused due to unusual development of skin cells and deadly type cancer. Early diagnosis is very significant and can avoid some categories of skin cancers, such as melanoma and focal cell carcinoma. The recognition and the classification of skin malignant growth in the beginning time is expensive and challenging. The deep learning architectures such as recurrent networks and convolutional neural networks (ConvNets) are developed in the past, which are proven appropriate for non-handcrafted extraction of complex features.

4.1.1 DISADVANTAGES OF EXISTING SYSTEM:

1. The recognition and the classification of skin malignant growth in the beginning time is expensive and challenging.
2. The deep learning architectures such as recurrent networks and convolutional neural networks (ConvNets) are developed in the past, which are proven appropriate for non-handcrafted extraction of complex features.

4.2 Proposed System:

The purpose of this study is to design a computer based melanoma lesion detection scheme that supports dermatologist as a decision support for melanoma classification. This paper provide a mechanism of feature fusion and suggests a classification framework by integrating ConvNet model with hand-crafted features as a cascaded ensembled model.

4.2.1 Advantages of proposed system:

1. the efficiency of the ConvNet models, a cascaded ensembled network that uses an integration of ConvNet and handcrafted features based multi-layer perceptron
2. It is demonstrated that accuracy of ensembled deep learning model is improved.

4.3 FUNCTIONAL REQUIREMENTS

1.Data Collection

2.Data Preprocessing

3.Training And Testing

4.Modiling

5.Predicting

4.4 NON FUNCTIONAL REQUIREMENTS

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system. Example of nonfunctional requirement, *"how fast does the website load?"* Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

- Usability requirement
- Serviceability requirement
- Manageability requirement
- Recoverability requirement
- Security requirement
- Data Integrity requirement
- Capacity requirement
- Availability requirement
- Scalability requirement
- Interoperability requirement
- Reliability requirement
- Maintainability requirement
- Regulatory requirement
- Environmental requirement

SYSTEM DESIGN

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE:

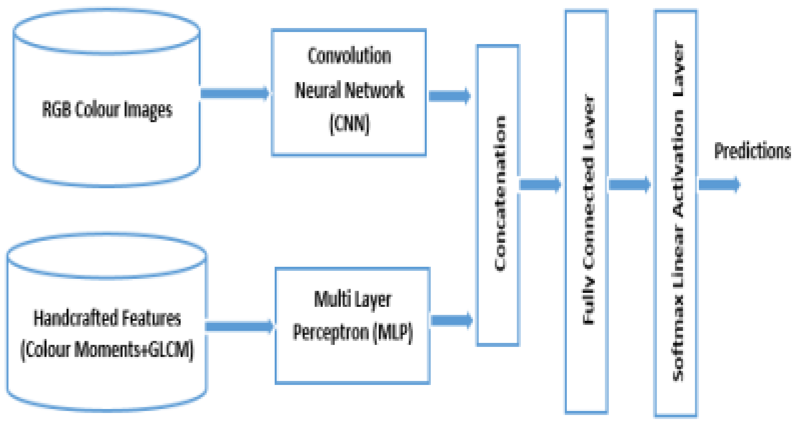

RGB Colour Images
Convolution Neural Network (CNN)
Handcrafted Features (Colour Moments+GLCM)
Multi Layer Perceptron (MLP)
Concatenation
Fully Connected Layer
Softmax Linear Activation Layer
Predictions

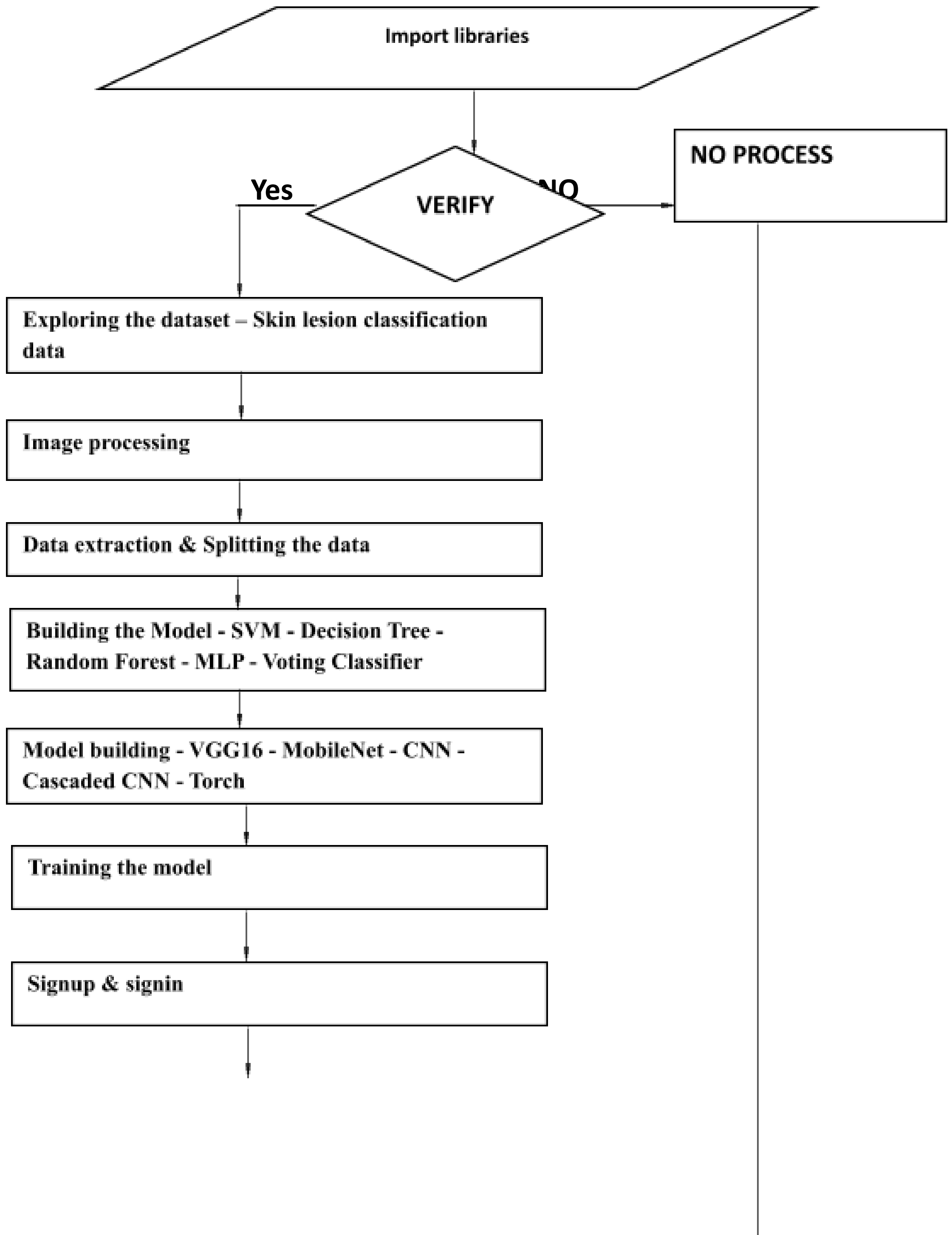
Fig.5.1.1 System architecture

DATA FLOW DIAGRAM:

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system,

various processing carried out on this data, and the output data is generated by this system.

2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.


Import libraries
VERIFY
Yes
NO
NO PROCESS
Exploring the dataset – Skin lesion classification data
Image processing
Data extraction & Splitting the data
Building the Model - SVM - Decision Tree - Random Forest - MLP - Voting Classifier
Model building - VGG16 - MobileNet - CNN - Cascaded CNN - Torch
Training the model
Signup & signin

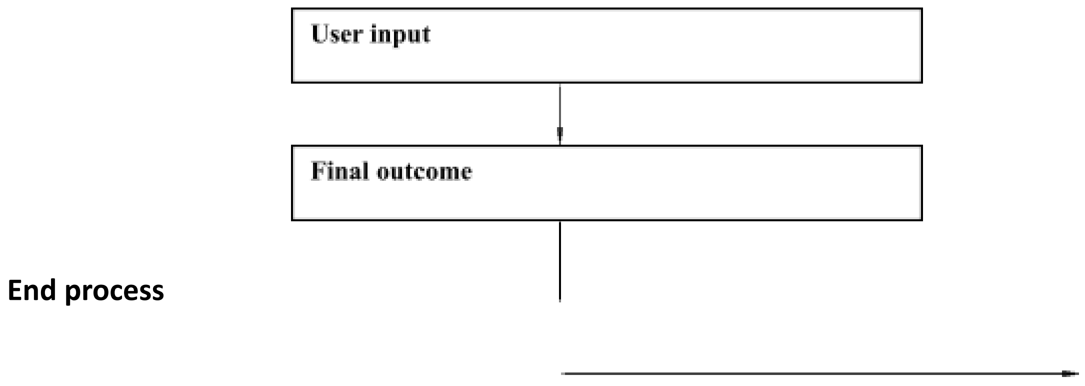


Fig.5.1.3 Dataflow diagram

5.2 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

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

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

Use case diagram:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

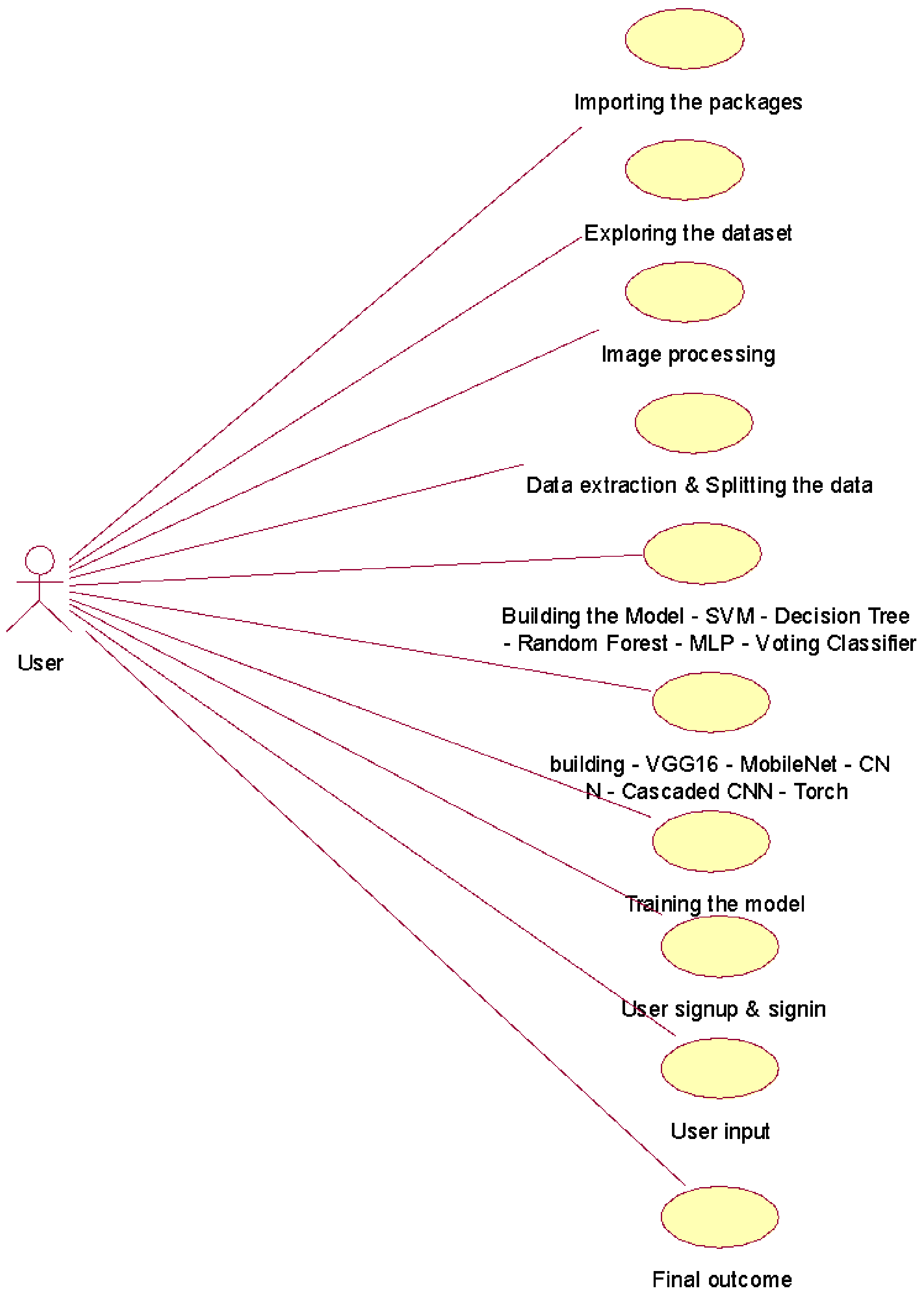

Importing the packages
Exploring the dataset
Image processing
Data extraction & Splitting the data
Building the Model - SVM - Decision Tree - Random Forest - MLP - Voting Classifier
building - VGG16 - MobileNet - CNN - Cascaded CNN - Torch
Training the model
User signup & signin
User input
Final outcome
User

Fig.5.2.1 Usecase diagram

Class diagram:

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

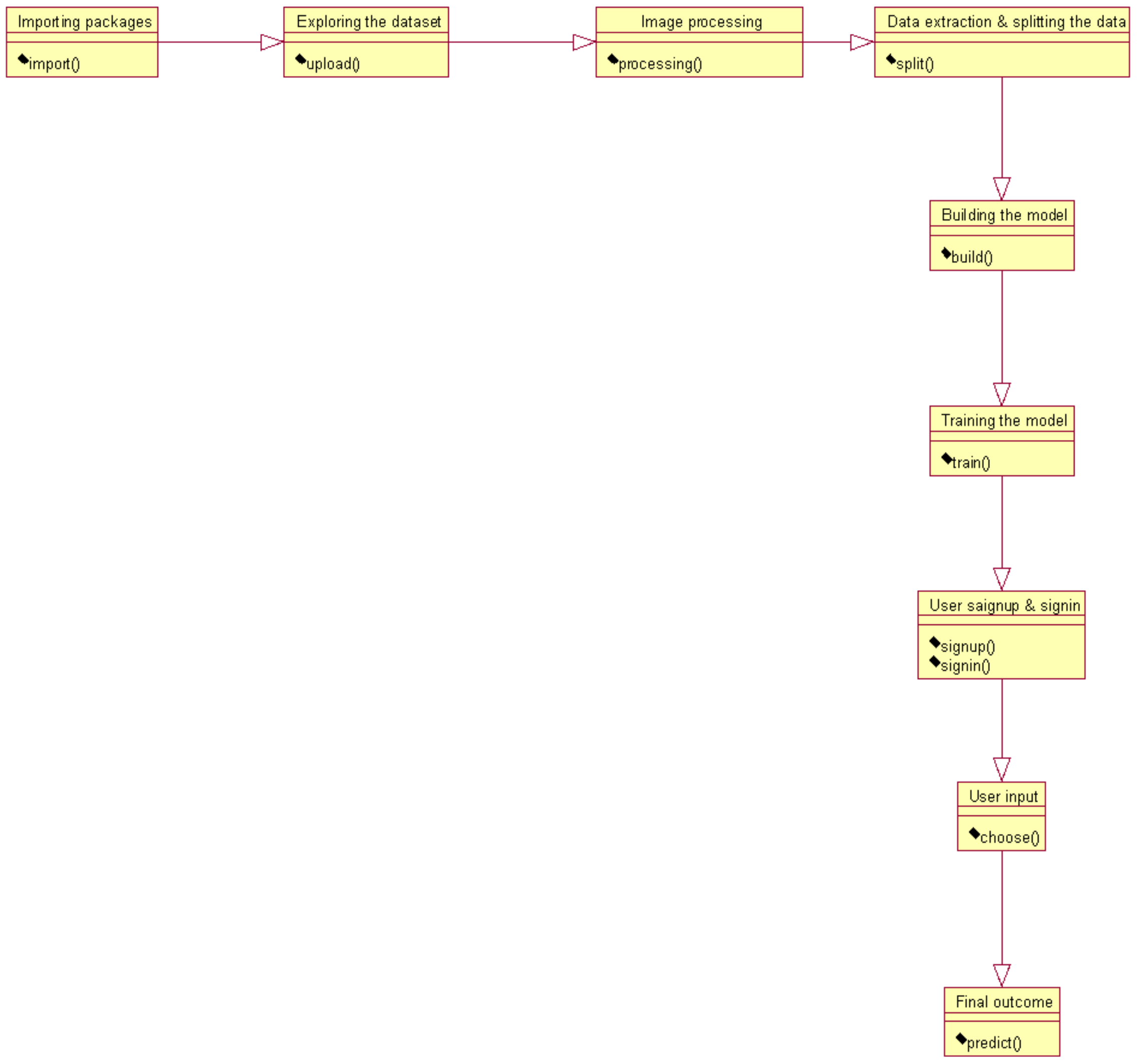


Fig.5.2.2 Class diagram

Activity diagram:

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.

Sequence diagram:

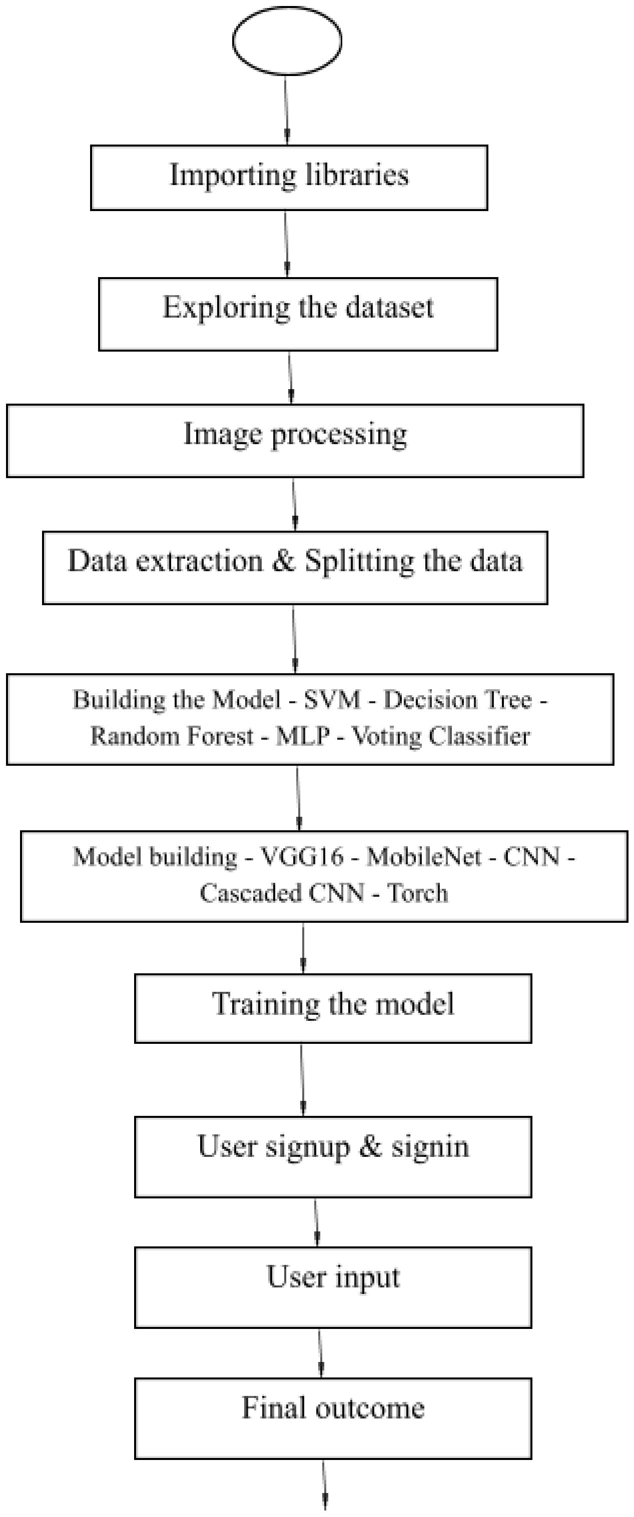

Importing libraries
Exploring the dataset
Image processing
Data extraction & Splitting the data
Building the Model - SVM - Decision Tree - Random Forest - MLP - Voting Classifier
Model building - VGG16 - MobileNet - CNN - Cascaded CNN - Torch
Training the model
User signup & signin
User input
Final outcome

Fig.5.2.3 Activity diagram

Sequence diagram:]

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

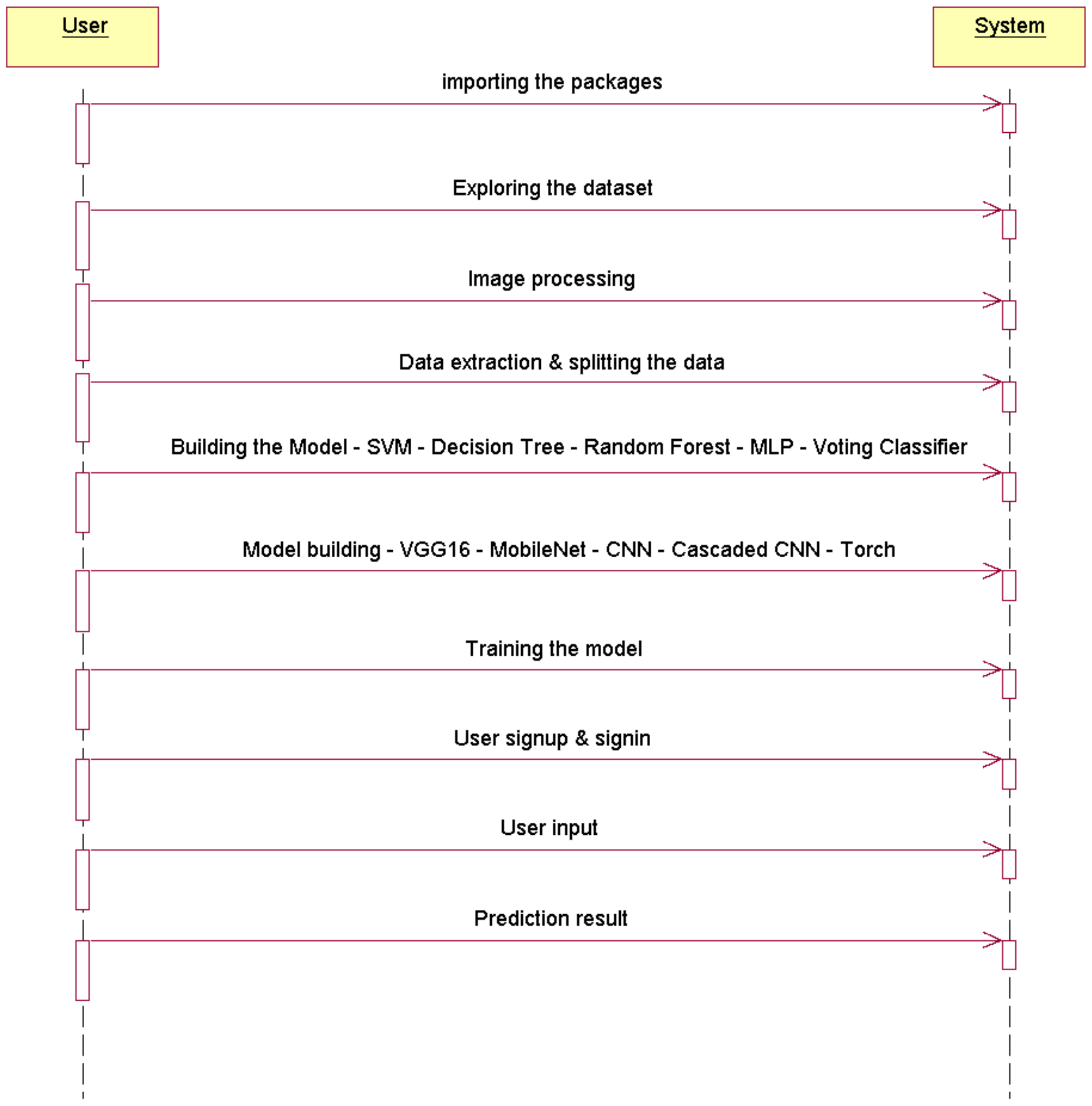

User
System
importing the packages
Exploring the dataset
Image processing
Data extraction & splitting the data
Building the Model - SVM - Decision Tree - Random Forest - MLP - Voting Classifier
Model building - VGG16 - MobileNet - CNN - Cascaded CNN - Torch
Training the model
User signup & signin
User input
Prediction result

Fig.5.2.4 Sequence diagram

Collaboration diagram:

A collaboration diagram groups together the interactions between different objects. The interactions are listed as numbered interactions that help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with other objects.

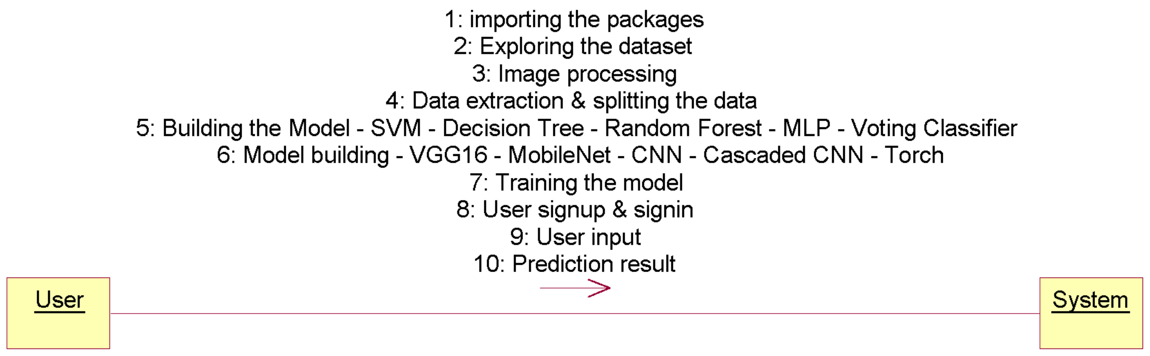

1: importing the packages
2: Exploring the dataset
3: Image processing
4: Data extraction & splitting the data
5: Building the Model - SVM - Decision Tree - Random Forest - MLP - Voting Classifier
6: Model building - VGG16 - MobileNet - CNN - Cascaded CNN - Torch
7: Training the model
8: User signup & signin
9: User input
10: Prediction result
User
System

Fig.5.2.5 Collaboration diagram

Component diagram:

The component diagram represents the high-level parts that make up the system. This diagram depicts, at a high level, what components form part of the system and how they are interrelated. A component diagram depicts the components culled after the system has undergone the development or construction phase.

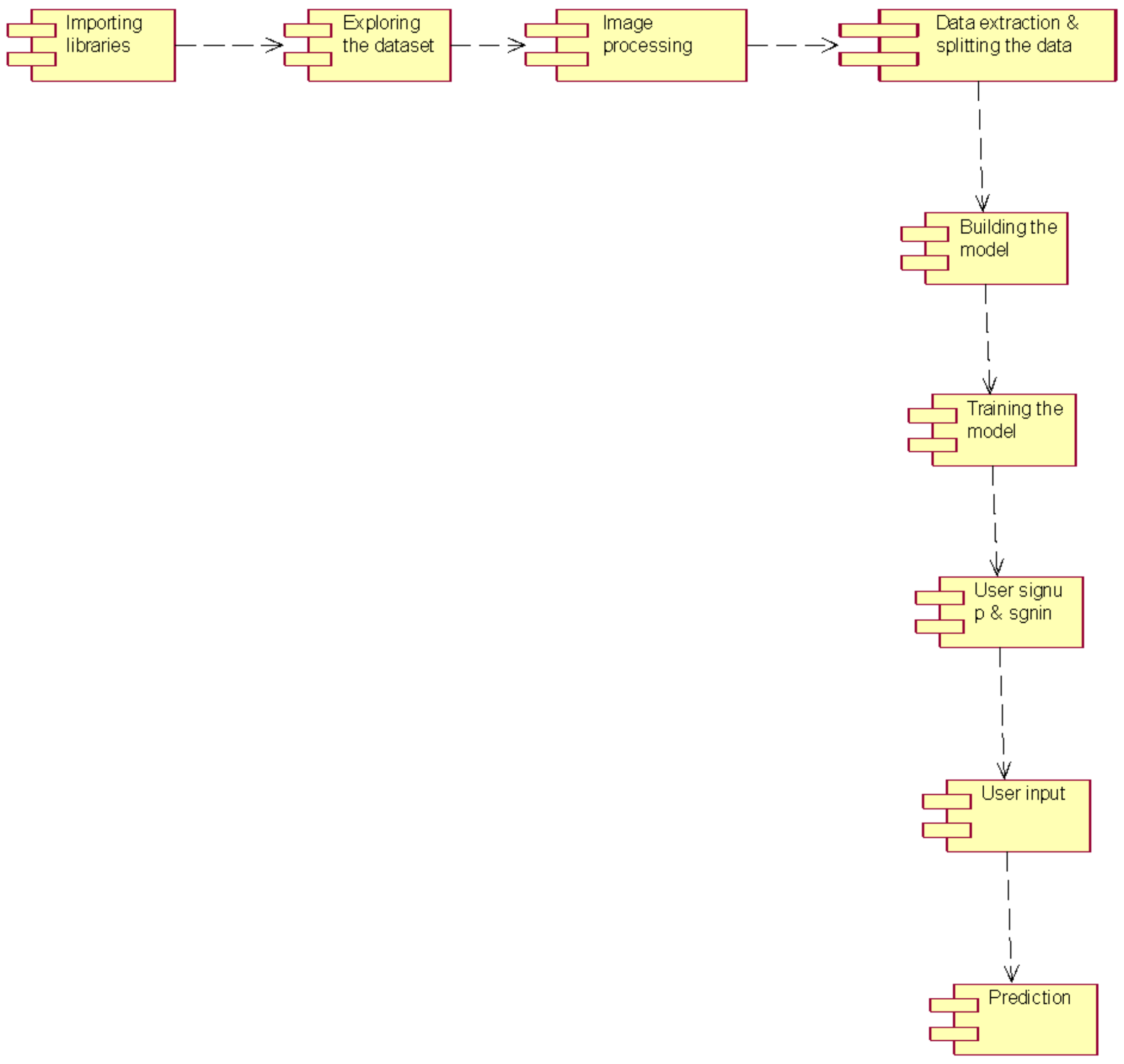


Fig.5.2.6 Component diagram

Deployment diagram:

The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far most useful when a system is built and ready to be deployed.

User
System

Fig.5.2.7 Deployment diagram

IMPLEMENTATION

6. IMPLEMENTATION

MODULES:

1. importing the packages: using this module we will import all packages
2. exploring the dataset – Skin lesion classification data: Using this module we will upload dataset
3. Image processing & cleaning: Using this module we will Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
4. Data extraction & Splitting the data: Using this module will divide dataset into train & test for processing
5. building the model: Using this module we will build all algorithms

   - SVM

   - Decision Tree

   - Random Forest

   - MLP

   - Voting Classifier

6. Image processing using Image Data Augmentation for CNN

7. Data Extraction to Train and Test

8. Model building

   - VGG16

   - MobileNet

- CNN

- Cascaded CNN - Torch

9. training the model: Using this module algorithms trained for processing & prediction
   MobileNet gives better accuracy aroubd 90% comparing with others propsed models only 5 epochs
10. Flask Framework with Sqlite for signup and signin: Using this module user will get register & login
    importing the packages
11. User gives input as Feature Values : Uisng this module user gives input for prediction
    the given input is preprocessed for prediction
12. trained model is used for prediction: Using this module predicted result displayed
    final outcome is displayed through frontend

Note:

Extension - VGG16 and MobileNet model is used as extension along with RF, Voting, SVM, DT

where MobileNet model gives better accuarcy comparing all other model and used to predicting the user inputs for analysis

ALGORITHMS:

a. Voting Classifier: A voting classifier is a machine learning estimator that trains various base models or estimators and predicts on the basis of aggregating the findings of each base estimator. The aggregating criteria can be combined decision of voting for each estimator output.

b. CNN: A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice.
c. SVM: Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.
d. DECISION TREE: A decision tree is a graph that uses a branching method to illustrate every possible output for a specific input. Decision trees can be drawn by hand or created with a graphics program or specialized software. Informally, decision trees are useful for focusing discussion when a group must make a decision.
e. RANDOM FOREST: Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.
f. MLP: Multi layer perceptron (MLP) is a supplement of feed forward neural network. It consists of three types of layers—the input layer, output layer and hidden layer, as shown in Fig. 3. The input layer receives the input signal to be processed.

g. VGG16: VGG16 is object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with 92.7% accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning.

h. MOBILENET: MobileNet model is a network model using depthwise separable convolution as its basic unit. Its depthwise separable convolution has two layers: depthwise convolution and point convolution.

6.2 SAMPLE CODE:

```
# Load
Librarie
s
            from glob import glob
            from functools import reduce
            import os
            import sys
            import yaml
            import shutil
            from tqdm import tqdm
            import pandas as pd
            import pprint

            import utils
            import pre_train
            from model_param import model_parameter

            from tensorflow.keras.applications import EfficientNetB3, EfficientNetB4,
            EfficientNetB5, EfficientNetB6, EfficientNetB7
            from tensorflow.keras import layers
            from tensorflow import keras
            from tensorflow.keras.preprocessing.image import ImageDataGenerator
            from tensorflow.keras.optimizers import Adam
            from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, CSVLogger
            from tensorflow.python.keras.utils.data_utils import Sequence
            from tensorflow.keras import backend as K

            import matplotlib.pyplot as plt
            from sklearn.metrics import accuracy_score
            import numpy as np

            import albumentations as A
            from ImageDataAugmentor.image_data_augmentor import ImageDataAugmentor
            from azureml.core import Workspace, Dataset
            from azureml.core.compute import ComputeTarget, ComputeInstance

            if __name__ == '__main__':
                # Load Azure subscription Detail
                # Subscription detail for Instance one
                azure_auth_stream = open("Azure_outh_settings_INSTANCE_1.yml", 'r')
                # Subscription detail for Instance two
                #azure_auth_stream = open("Azure_outh_settings_INSTANCE_2.yml", 'r')

                azure_settings = yaml.load(azure_auth_stream, yaml.SafeLoader)

                ## Change working Directory
                os.chdir('../')
                print("\nCurrent Working Directory: ", os.getcwd())

                # Azure subscription detail
                subscription_id = azure_settings['subscription_id']
                resource_group = azure_settings['resource_group']
                workspace_name = azure_settings['workspace_name']

                workspace = Workspace(subscription_id, resource_group, workspace_name)
                print("\nAzure Workspace Name: ", workspace.name)
                print("Azure Workspace Resource Group: ", workspace.resource_group)

                # Initialise Azure Instance
                try:
                    instance = ComputeInstance(workspace = workspace, name =
            azure_settings['instance_name'])
                    # Get Status
                    print('Azure ML Instance is {}.\n'.format(instance.get_status().state))
                except:
                    print("An exception occurred while initialising the Azure ML Compute")
                    sys.exit()

                # Select model to train
                # We have model2, model10, model16 and model12 model available
                model_config = 'model10'

                # Get the model parameters
                selected_model = model_parameter(model_config)

                # Define Model Log and Plot files
                log_path = "./runs/"
                os.makedirs(log_path, exist_ok = True)

                plot_path = "./plot/"
                os.makedirs(plot_path, exist_ok = True)

                ##################### Define path for Training and Testing Data
                train_path = "./{0}x{0}/".format(selected_model['input_image_size'])
                test_path = "./{0}x{0}_test/".format(selected_model['input_image_size'])
                save_model_path = "./saveModel/"
                os.makedirs(save_model_path, exist_ok = True)

                ##################### Get Training and Testing Labels from Azure Instance
                train_label = Dataset.get_by_name(workspace,
            name='train_2020_and_2019_with_9_Labels')
                test_label = Dataset.get_by_name(workspace, name='test_2020_no_PateintDetail')

                label = train_label.to_pandas_dataframe()
                test_csv = test_label.to_pandas_dataframe()

                # Append Image extension and file path to train and test CSV
                absolute_path_train = os.path.abspath(train_path)
                label = utils.append_path(label, absolute_path_train)

                absolute_path_test = os.path.abspath(test_path)
                test_csv = utils.append_path(test_csv, absolute_path_test)

                ##################### Hyper Parameter
                hyper_param = {
                    'seed': 42,
                    'image_size': selected_model['resize'], # resize image
                    'backbone_model': selected_model['backbone'], # Pretrained model name
                    'early_stop': 10,
                    'num_class': selected_model['target'],
                    'train_batch_size': selected_model['train_batch_size'], # Train Batch Size
                    'test_batch_size': 1, # Testing set batch size
                    'validation_batch_size': selected_model['validation_batch_size'], #
            Validation Batch Size
                    'epoch': selected_model['epochs'],
                    'warmup_epoch': 1,
                    'learning_rate_base': selected_model['initial_lr'], # Base learning rate
            after warmup.
                    'warmup_learning_rate': selected_model['initial_lr'], # Warmup learning
            rate
                    'training_sample_count': label.shape[0], # Number of training sample
                    'save_model': selected_model['savedModelByName'], # save model name
                    'save_final_model': selected_model['saveFinalModelBy'] # Save final
            trained model in Tensorflow Format
                }

                image_resize = (hyper_param['image_size'], hyper_param['image_size'])
                image_shape = image_resize + (3, )
                # Total training steps in Warmup
                total_steps = int(hyper_param['epoch'] * hyper_param['training_sample_count']
            / hyper_param['train_batch_size'])
                # Compute the number of warmup batches.
                warmup_steps = int(hyper_param['warmup_epoch'] *
            hyper_param['training_sample_count'] / hyper_param['train_batch_size'])

                # Print Hyper parameter
                if selected_model['print_hyper_parameter']:
                    print("\n####################### Hyper Parameter 
            #################################\n")
                    pprint.pprint(hyper_param)

                    print('\nImage Shape: {}'.format(image_shape))
                    print('Total training steps in Warmup: {}'.format(total_steps))
                    print('Number of Warmup Batch: {}\n'.format(warmup_steps))
                    print("\nTrain Label shape: ", label.shape)
                    print("Test Label shape: ", test_csv.shape)

                ########################### Train model
                # Create the Learning rate scheduler.
                warm_up_lr = utils.WarmUpCosineDecayScheduler(learning_rate_base = 
            hyper_param['learning_rate_base'],
                                                        total_steps = total_steps,
                                                        warmup_learning_rate = 
            hyper_param['warmup_learning_rate'],
                                                        warmup_steps = warmup_steps,
                                                        hold_base_rate_steps = 0)

                # Initialise Pre-train Model
                model = pre_train.EffNet(input_size = image_shape, num_classess = 
            hyper_param['num_class'], \
                    pretrained_model = hyper_param['backbone_model'], \
                    lr_rate = hyper_param['learning_rate_base'], \
                    print_trainable_layers = selected_model['print_trainable_layers'],\
                    print_model_summary = selected_model['print_model_summary'])

                # Preprocess and Augment Image for train, test and validation set.
                transform_train, transform_val, transform_test = \
                    pre_train.augment_images(hyper_param['image_size'])

                # Prepare train, validation Generator
                train_generator, validation_generator = pre_train.data_generator(seed = 
            hyper_param['seed'],\
                    transforms_train = transform_train, transforms_val = transform_val, label 
            = label, \
                    train_path = train_path, image_resize = image_resize,  train_batch_size = 
            selected_model['train_batch_size'], \
                    validation_batch_size = selected_model['validation_batch_size'])

                # Visualise preprocess and augmented data
                if selected_model['visualise_augmented_data']:
                    # Get Train set
                    train_generator.show_data()
                    # Get validation set
                    validation_generator.show_data()

                ## Define Callbacks
                # Define Early Stopping on validation loss
                es = EarlyStopping(monitor='val_loss', mode = 'min', patience = 
            hyper_param['early_stop'],\
                    verbose = 1, restore_best_weights = True)
                # Save model after each epoch
                ck = ModelCheckpoint(save_model_path + hyper_param['save_model'],
            monitor='val_loss', \
                    verbose = 1, save_best_only = False, save_weights_only= False,
            mode='auto')

                # Save logs to CSV
                # append=False -> overwrite existing file.
                logs = CSVLogger(log_path + selected_model['log_by'], separator=",",
            append=False)

                # Callback list
                call_backs = [warm_up_lr, ck, logs]

                # Get Train and validation step size
                STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
                STEP_SIZE_VALID = validation_generator.n//validation_generator.batch_size

                start_training = input("Do you want to start training the model? [y]es OR
            [n]o: ")
                # Start the training process is the 'yes' input is received from the terminal
                if start_training == 'y':
                    print('\n\n---------------- Staring the Training Process...
            --------------- ')
                    # Train the model
                    history = pre_train.train_model(model = model, train_generator =
            train_generator, epoch = hyper_param['epoch'], \
                        train_batch_size = hyper_param['train_batch_size'],
            validation_generator = validation_generator, \
                        validation_batch_size = hyper_param['validation_batch_size'],
            train_step = STEP_SIZE_TRAIN, \
                        valid_step = STEP_SIZE_VALID, callback = call_backs)
                    print("\n ----------------- Model is trained --------------------------")
                else:
                    print("Training is cancelled.....\nTerminating Python...")
                    sys.exit()

                # Plot Training and validation loss
                print("\n------ Saving Training and Validation Plot --------")
                # Training and validation: accuracy & loss
                utils.save_plot(history = history, \
                    save_dir = plot_path + selected_model['save_plot_name'])
```

SOFTWARE ENVIRONMENT

7.SOFTWARE ENVIRONMENT

PYTHON LANGUAGE:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python is a dynamic, high-level, free open source, and interpreted programming language. It supports object-oriented programming as well as procedural-oriented programming. In Python, we don’t need to declare the type of variable because it is a dynamically typed language. For example, x = 10 Here, x can be anything such as String, int, etc.

Features in Python:

There are many features in Python, some of which are discussed below as follows:

1. Free and Open Source

Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

2. Easy to code

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

3. Easy to Read

As you will see, learning Python is quite simple. As was already established, Python's syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

4. Object-Oriented Language

One of the key features of Python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

5. GUI Programming Support

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

6. High-Level Language

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

7. Extensible feature

Python is an Extensible language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

8. Easy to Debug

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program's issues once you understand how to interpret Python's error traces. Simply by glancing at the code, you can determine what it is designed to perform.

9. Python is a Portable language

Python language is also a portable language. For example, if we have Python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

10. Python is an Integrated language

Python is also an Integrated language because we can easily integrate Python with other languages like C, C++, etc.

11. Interpreted Language:

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called bytecode.

12. Large Standard Library

Python has a large standard library that provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in Python such as regular expressions, unit-testing, web browsers, etc.

13. Dynamically Typed Language

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

14. Frontend and backend development

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like javascript. Backend is the strong forte of Python it's extensively used for this work cause of its frameworks like Django and Flask.

15. Allocating Memory Dynamically

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

LIBRARIES/PACKGES :-

Tensorflow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

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 tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards 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.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

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.

SYSTEM TESTING

8.SYSTEM TESTING

System testing, also referred to as system-level tests or system-integration testing, is the process in which a quality assurance (QA) team evaluates how the various components of an application interact together in the full, integrated system or application. System testing verifies that an application performs tasks as designed. This step, a kind of black box testing, focuses on the functionality of an application. System testing, for example, might check that every kind of user input produces the intended output across the application.

Phases of system testing:

A video tutorial about this test level. System testing examines every component of an application to make sure that they work as a complete and unified whole. A QA team typically conducts system testing after it checks individual modules with functional or user-story testing and then each component through integration testing.

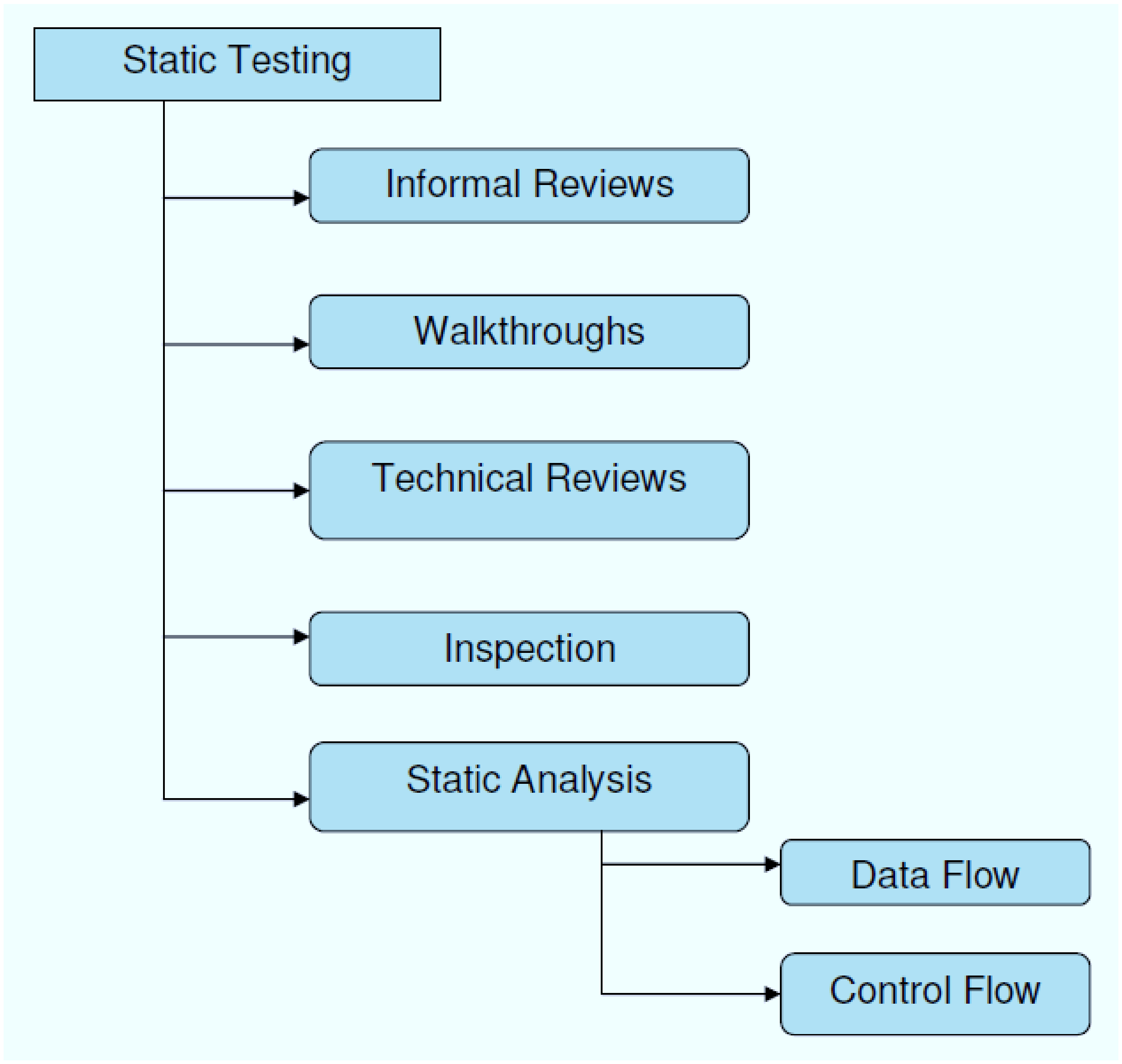
If a software build achieves the desired results in system testing, it gets a final check via acceptance testing before it goes to production, where users consume the software. An app-dev team logs all defects, and establishes what kinds and amount of defects are tolerable.

8.1Software Testing Strategies:

Optimization of the approach to testing in software engineering is the best way to make it effective. A software testing strategy defines what, when, and how to do whatever is necessary to make an end-product of high quality. Usually, the following software testing strategies and their combinations are used to achieve this major objective:

Static Testing:

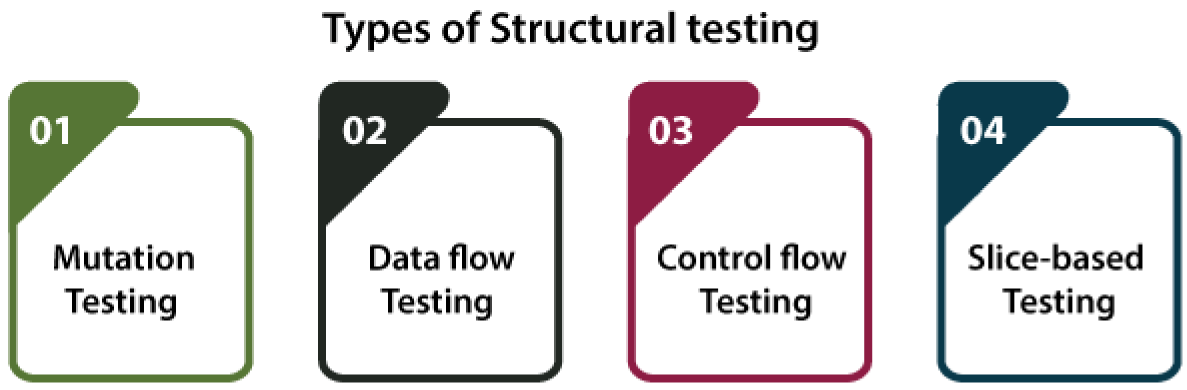
The early-stage testing strategy is static testing: it is performed without actually running the developing product. Basically, such desk-checking is required to detect bugs and issues that are present in the code itself. Such a check-up is important at the pre-deployment stage as it helps avoid problems caused by errors in the code and software structure deficits.



Structural Testing:

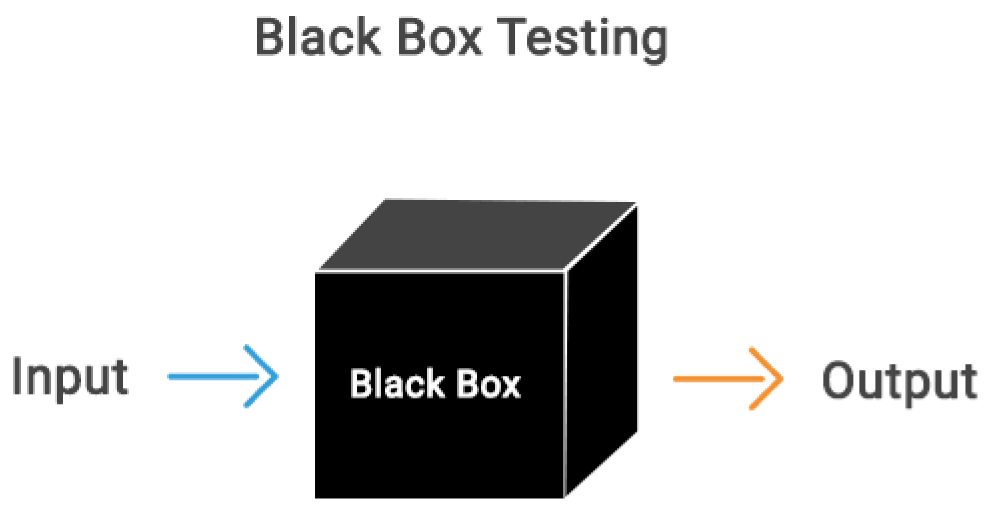
It is not possible to effectively test software without running it. Structural testing, also known as white-box testing, is required to detect and fix bugs and errors emerging during the pre-production stage of the software development process. At this stage, unit testing based on the software structure is performed using regression testing. In most cases, it is an automated process working within the test automation framework to speed up the development process at this stage.

Developers and QA engineers have full access to the software's structure and data flows (data flows testing), so they could track any changes (mutation testing) in the system's behavior by comparing the tests' outcomes with the results of previous iterations (control flow testing).



Behavioral Testing:

The final stage of testing focuses on the software's reactions to various activities rather than on the mechanisms behind these reactions. In other words, behavioral testing, also known as black-box testing, presupposes running numerous tests, mostly manual, to see the product from the user's point of view. QA engineers usually have some specific information about a business or other purposes of the software ('the black box') to run usability tests, for example, and react to bugs as regular users of the product will do. Behavioral testing also may include automation (regression tests) to eliminate human error if repetitive activities are required. For example, you may need to fill 100 registration forms on the website to see how the product copes with such an activity, so the automation of this test is preferable.

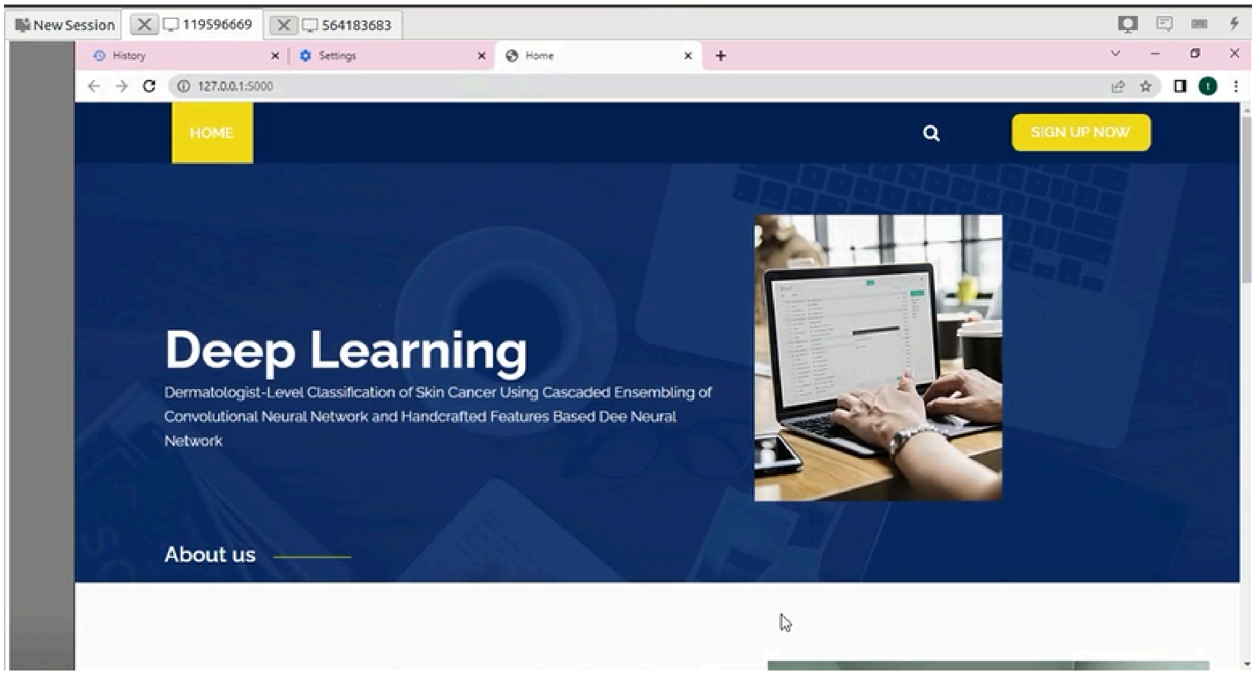


8.2 TEST CASES:

S.NO	INPUT	If available	If not available
1	User signup	User get registered into the application	There is no process
2	User signin	User get login into the application	There is no process
3	Enter input for prediction	Prediction result displayed	There is no process

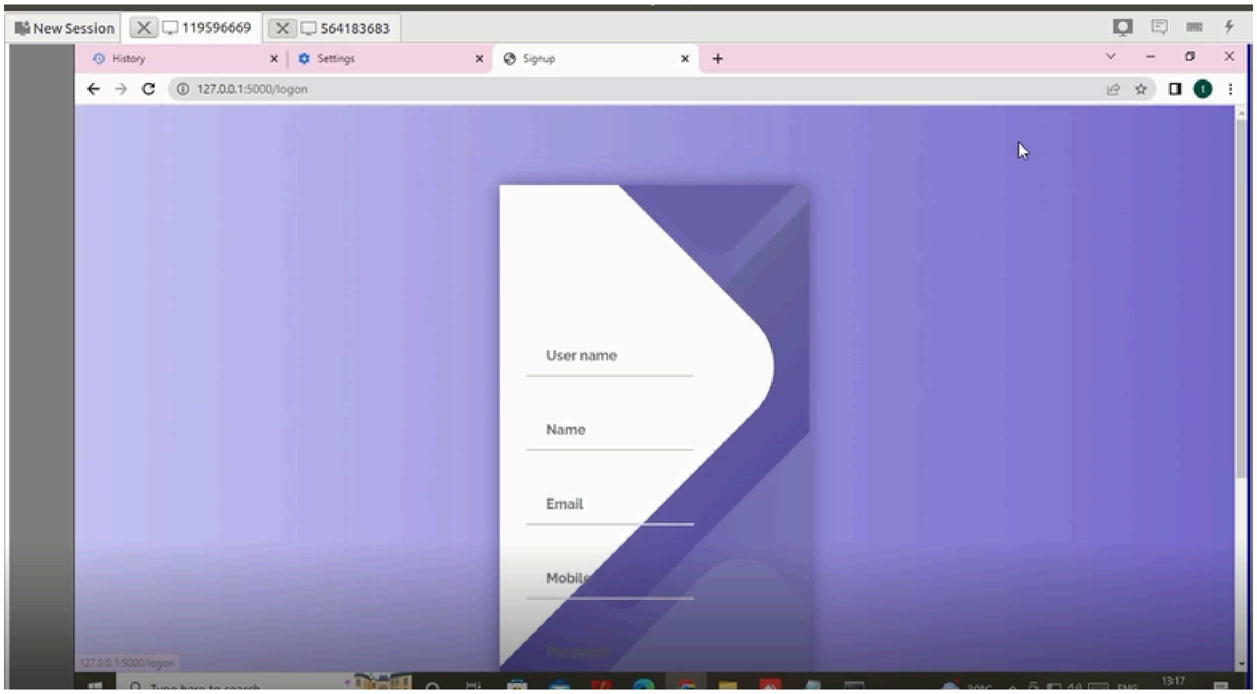
SCREENS

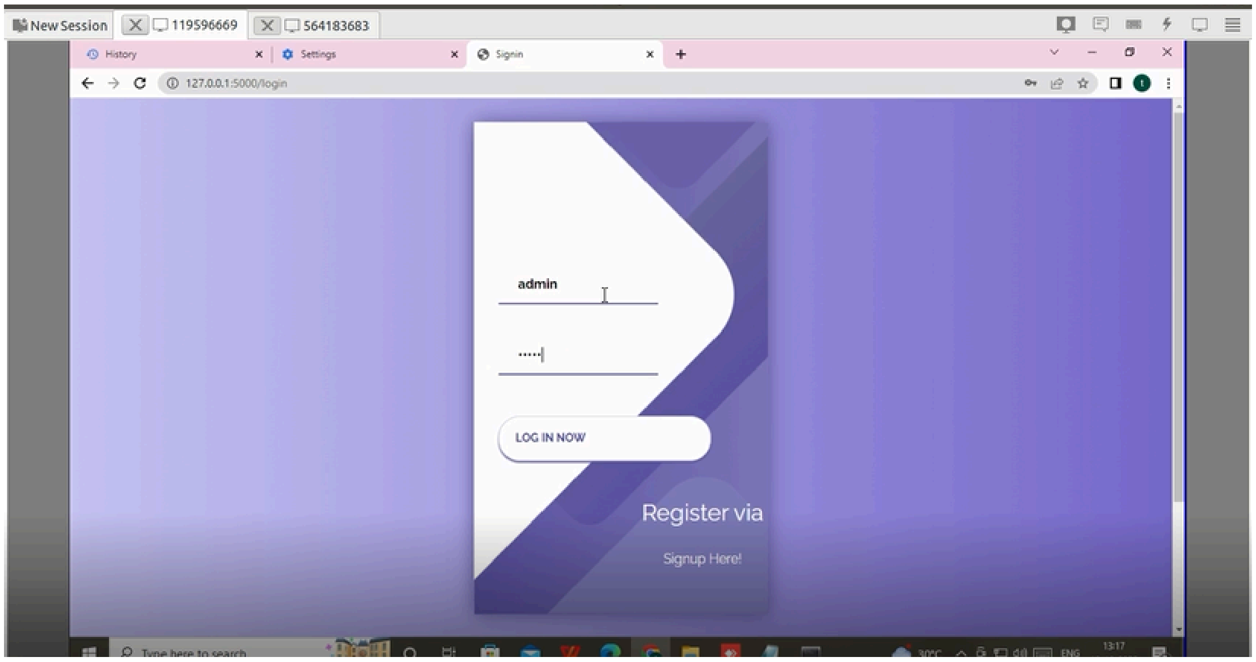
7. SCREENSHOTS

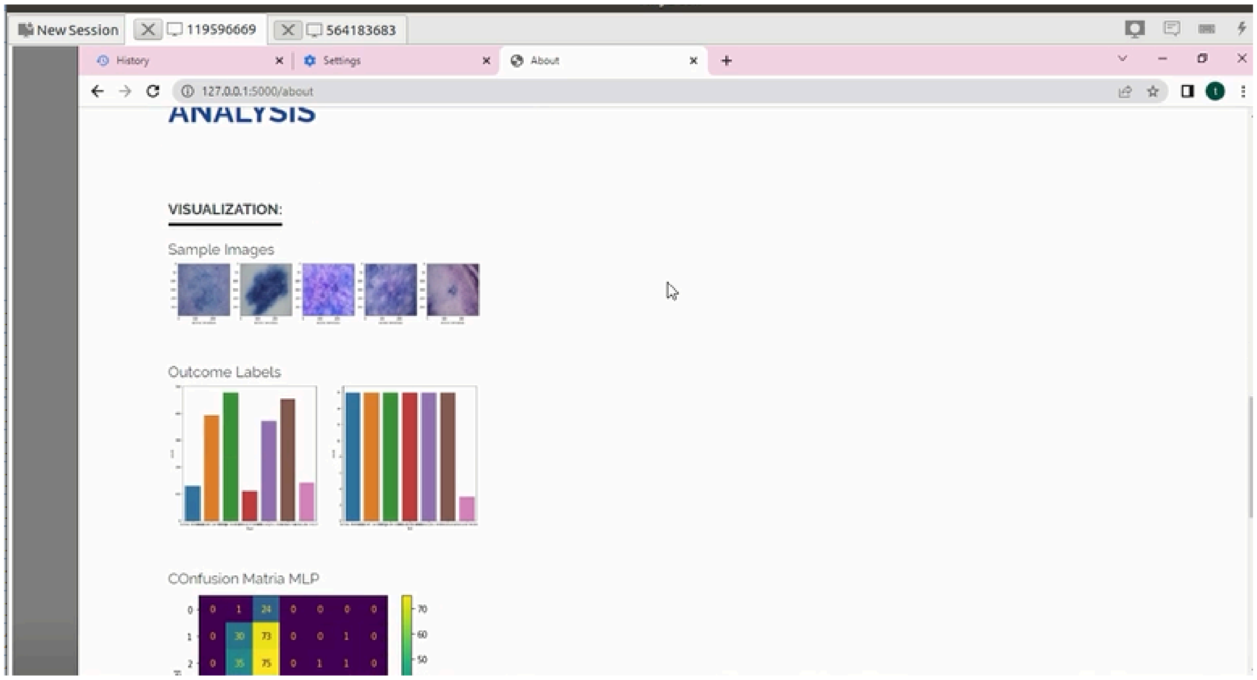


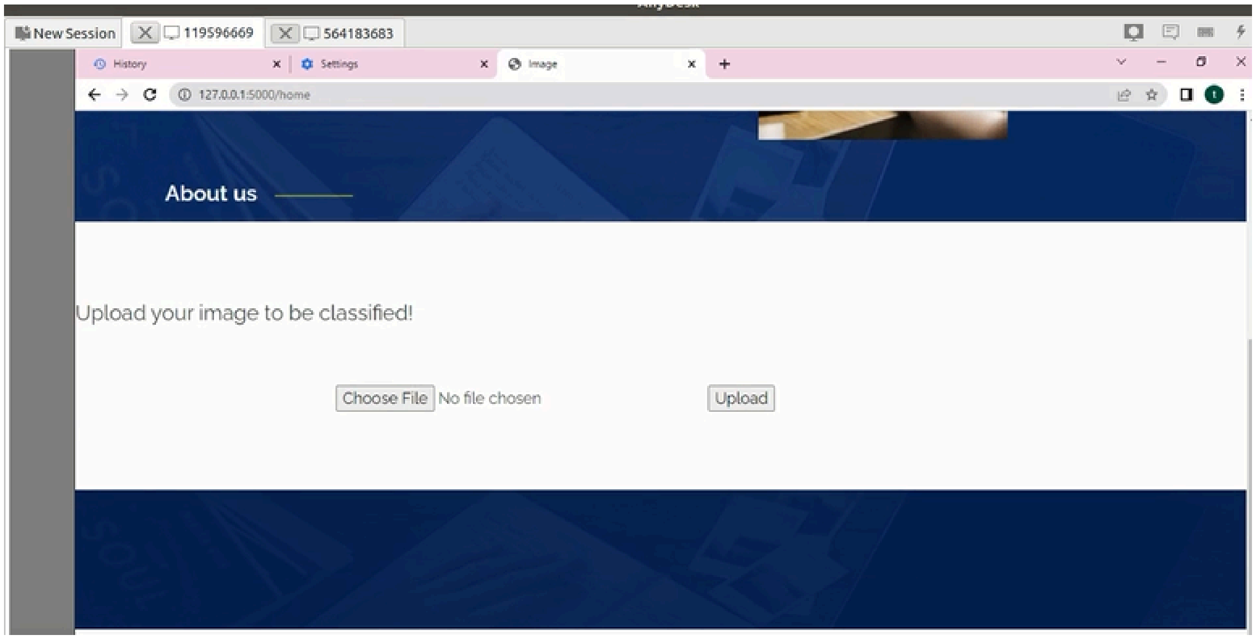
Deep Learning

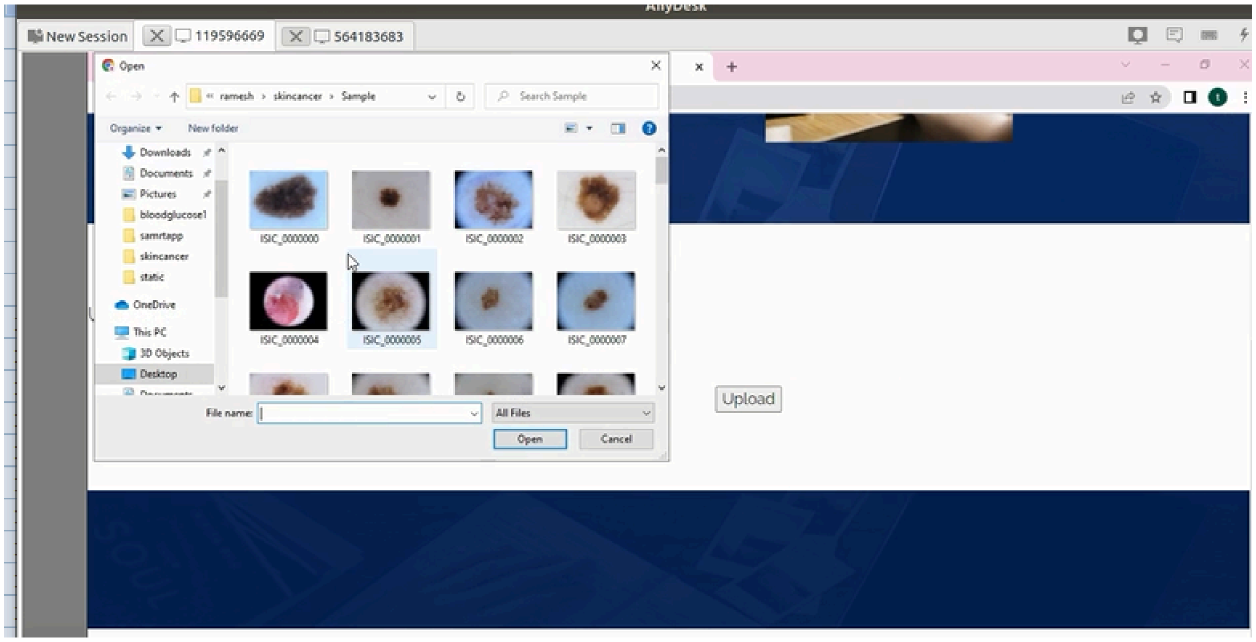
Dermatologist-Level Classification of Skin Cancer Using Cascaded Ensembling of Convolutional Neural Network and Handcrafted Features Based Dee Neural Network

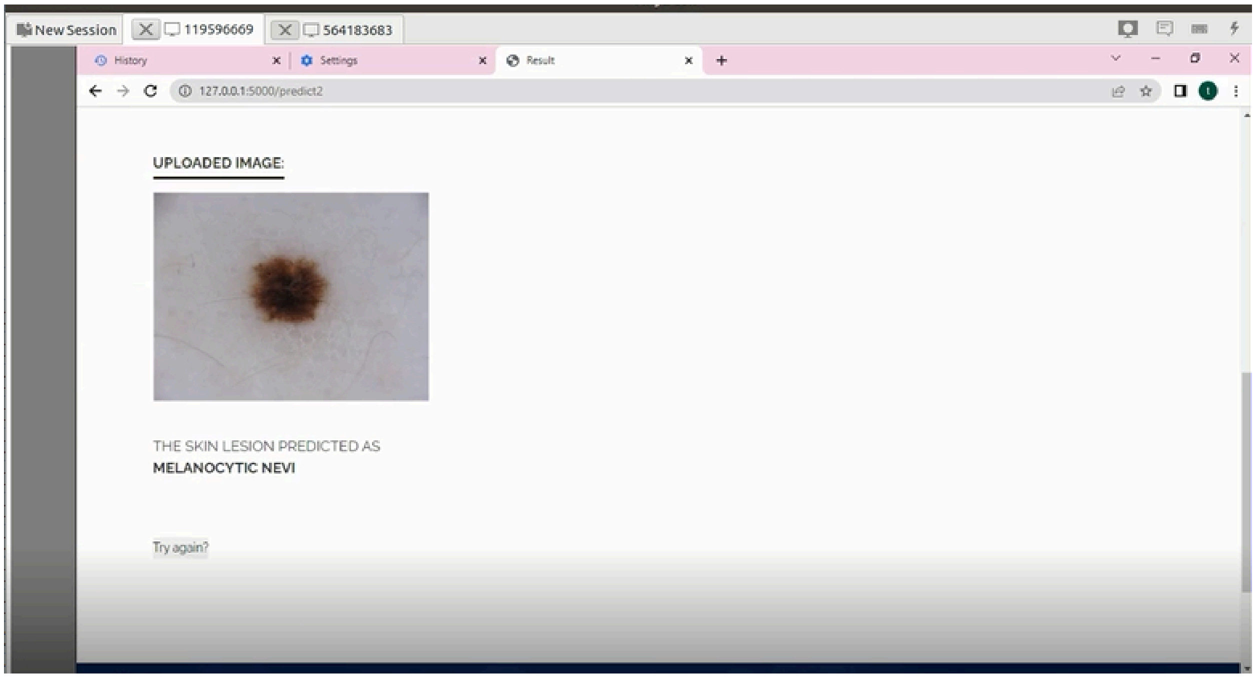



New Session
119596669
564183683
History
Settings
Signin
127.0.0.1:5000/login
admin
LOG IN NOW
Register via
Signup Here!


New Session
119596669
564183683
History
Settings
About
127.0.0.1:5000/about
ANALYSIS
VISUALIZATION:
Sample Images
Outcome Labels
COnfusion Matria MLP


New Session
119596669
564183683
History
Settings
Image
127.0.0.1:5000/home
About us
Upload your image to be classified!
Choose File
No file chosen
Upload

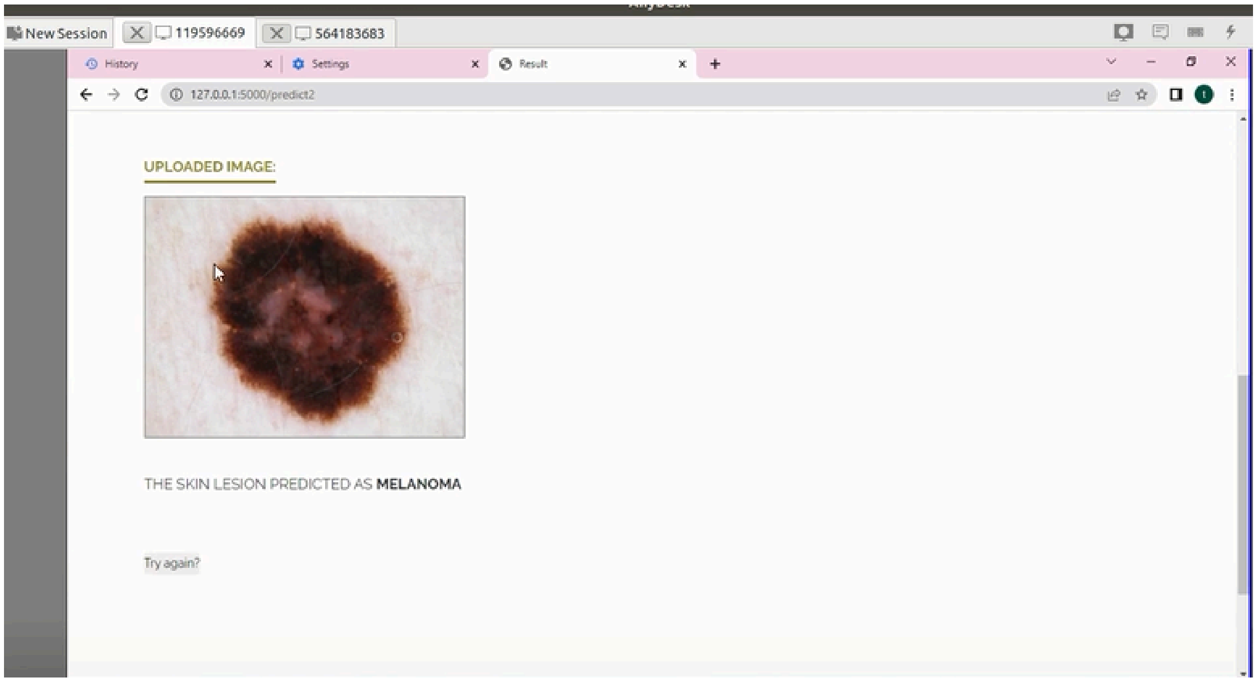

New Session
119596669
564183683
Open
ramesh › skincancer › Sample
Search Sample
Organize
New folder
Downloads
Documents
Pictures
bloodglucose1
samrtapp
skincancer
static
OneDrive
This PC
3D Objects
Desktop
ISIC_0000000
ISIC_0000001
ISIC_0000002
ISIC_0000003
ISIC_0000004
ISIC_0000005
ISIC_0000006
ISIC_0000007
File name:
All Files
Open
Cancel
Upload



UPLOADED IMAGE:

THE SKIN LESION PREDICTED AS
MELANOCYTIC NEVI

Try again?



UPLOADED IMAGE:

THE SKIN LESION PREDICTED AS **MELANOMA**

Try again?

CONCLUSION

10.CONCLUSION

Considering the current achievement of deep learning architectures, an efficient method is presented for the skin lesion classification. In this work, a cascaded model is created that combines the strengths of models based on hand-crafted feature extraction approaches and deep learning model. To gain the high accuracy of the skin disease image classification the powerful ability of feature learning of deep ConvNets is integrated with handcrafted features including colour moments and texture features. This deep learning architecture termed as cascaded ensembled deep learning model in this paper. The simulation results show that our proposed model outperforms the ConvNet model. More research is being done to create a more robust model by combining clinical features like sex, age, itching, burns, medical history, and location with handmade features to create a more robust model.

BIBILOGRAPHY

11. REFERENCES

[1] D. Didona, G. Paolino, U. Bottoni, and C. Cantisani, ‘‘Non melanoma skin cancer pathogenesis overview,’’ Biomedicines, vol. 6, no. 1, p. 6, Jan. 2018.

[2] T. J. Brinker, A. Hekler, J. S. Utikal, N. Grabe, D. Schadendorf, J. Klode, C. Berking, T. Steeb, A. H. Enk, and C. von Kalle, ‘‘Skin cancer classification using convolutional neural networks: Systematic review,’’ J. Med. Internet Res., vol. 20, no. 10, Oct. 2018, Art. no. e11936.

[3] P. Tschandl, C. Rinner, Z. Apalla, G. Argenziano, N. Codella, A. Halpern, M. Janda, A. Lallas, C. Longo, J. Malvehy, J. Paoli, S. Puig, C. Rosendahl, H. Peter Soyer, I. Zalaudek, and H. Kittle, ‘‘Human–computer collaboration for skin cancer recognition,’’ Nature Med. vol. 26, no. 8, pp. 1229–1234, 2020.

[4] B. C. Baumann, K. M. MacArthur, J. D. Brewer, W. M. Mendenhall, C. A. Barker, J. R. Etzkorn, N. J. Jellinek, J. F. Scott, H. A. Gay, J. C. Baumann, F. A. Manian, P. M. Devlin, J. M. Michalski, N. Y. Lee, W. L. Thorstad, L. D. Wilson, C. A. Perez, and C. J. Miller, ‘‘Management of primary skin cancer during a pandemic: Multidisciplinary recommendations,’’ Cancer, vol. 126, no. 17, pp. 3900–3906, Sep. 2020.

[5] J. Höhn et al., ‘‘Combining CNN-based histologic whole slide image analysis and patient data to improve skin cancer classification,’’ Eur. J. Cancer, vol. 149, pp. 94–101, May 2021.

[6] C. A. Hartanto and A. Wibowo, ‘‘Development of mobile skin cancer detection using faster R-CNN and MobileNet v2 model,’’ in Proc. 7th Int. Conf. Inf. Technol., Comput., Electr. Eng. (ICITACEE), Sep. 2020, pp. 58–63.

[7] M. A. Kassem, K. M. Hosny, R. Dama†evi£ius, and M. M. Eltoukhy, ‘‘Machine learning and deep learning methods for skin lesion classification and diagnosis: A systematic review,’’ Diagnostics, vol. 11, no. 8, p. 1390, Jul. 2021.

[8] O. O. Abayomi-Alli, R. Dama†evi£ius, S. Misra, R. Maskeliunas, and A. Abayomi-Alli, ‘‘Malignant skin melanoma detection using image augmentation by oversampling in nonlinear lower-dimensional embedding manifold,’’ TURKISH J. Electr. Eng. Comput. Sci., vol. 29, nos. SI–1, pp. 2600–2614, Oct. 2021.

[9] T. C. Pham, V. D. Hoang, C. T. Tran, M. S. K. Luu, D. A. Mai, A. Doucet, and C. M. Luong, ‘‘Improving binary skin cancer classification based on best model selection method combined with optimizing full connected layers of deep CNN,’’ in Proc. Int. Conf. Multimedia Anal. Pattern Recognit. (MAPR), Oct. 2020, pp. 1–6.

[10] N. Hameed, A. Ruskin, K. Abu Hassan, and M. A. Hossain, ‘‘A comprehensive survey on image-based computer aided diagnosis systems for skin cancer,’’ in Proc. 10th Int. Conf. Softw., Knowl., Inf. Manage. Appl. (SKIMA), 2016, pp. 205–214.