

A Major Project Report On

“HUMAN ACTIVITY RECOGNITION”

Submitted to partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

T.SHARATH REDDY

20911A1250

UNDER THE GUIDANCE OF

MRS.K.S.PADMA PRIYA

ASSISTANT PROFESSOR

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY



(ACCREDITED BY NBA, APPROVED BY AICTE, AUTONOMOUS VJIT HYDERABAD)

AZIZ NAGAR GATE, C.B.POST, CHILKUR ROAD, HYDERABAD – 500075

2023 - 2024

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

(ACCREDITED BY NBA, APPROVED BY AICTE, AUTONOMOUS VJIT HYDERABAD)

AZIZ NAGAR GATE, C.B.POST, CHILKUR ROAD, HYDERABAD – 500075

DEPARTMENT OF INFORMATION TECHNOLOGY



CERTIFICATE

This is to certify that the project report titled “**HUMAN ACTIVITY RECOGNITION**” is a bonafide work by T. Sharath Chandra Reddy - 20911A1250 , in partial fulfillment for the award of the degree of Bachelor of Technology in “**INFORMATION TECHNOLOGY**”, VJIT Hyderabad during the year **2023 - 2024**.

Project Guide

Mrs.K.S.Padma Priya

MTech

Assistant Professor

Head of Department

Mr.B.Srinivasulu

M.E., (Ph.D)

Professor

External Examiner

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

(ACCREDITED BY NBA, APPROVED BY AICTE, AUTONOMOUS VJIT HYDERABAD)

AZIZ NAGAR GATE, C.B.POST, CHILKUR ROAD, HYDERABAD – 500075

2023 – 2024



DECLARATION

We, T.Sharath Chandra Reddy - 20911A1250, Here by declare that the project report entitled, “HUMAN ACTIVITY RECOGNITION” is submitted in the partial fulfillment of the requirement for the award of Bachelor of Technology in Information Technology to Vidya Jyothi Institute of Technology, Autonomous VJIT-Hyderabad, is authentic work and has not been submitted to any other university or institute for the degree.

T SHARATH REDDY (20911A1250)

ACKNOWLEDGEMENT

It is a great pleasure to express our deepest sense of gratitude and indebtedness to our internal guide **Mrs.K.S.Padma Priya**, Assistant Professor, Department of IT, VJIT, for having been a source of constant inspiration, precious guidance and generous assistance during the project work. We deem it as a privilege to have worked under her able guidance. Without her close monitoring and valuable suggestions this work would not have taken this shape. We feel that this help is unsubstitutable and unforgettable.

We wish to express our sincere thanks to **Dr. E. Saibaba Reddy**, Principal VJIT, for providing the college facilities for the completion of the project. We are profoundly thankful to **Mr.B.Srinivasulu**, Professor and Head of Department of IT, for his cooperation and encouragement. Finally, we thank all the faculty members, supporting staff of IT Department and friends for their kind co-operation and valuable help for completing the project.

T.SHARATH REDDY

(20911A1250)

ABSTRACT

Human Activity Recognition using deep learning involves the development and application of algorithms that can automatically identify and classify human activities based on labeled datasets. This field has gained significant attention due to its potential applications in various domains, including healthcare, sports, and security. The abstract concept of Human Activity Recognition using machine learning revolves around utilizing pre-existing datasets, extracting relevant features, and using machine learning techniques to build models that can accurately classify different activities. The process typically starts with the selection of appropriate datasets containing labeled samples of various human activities. These datasets provide a rich source of information for training machine learning models. The labeled samples associate specific activities with corresponding features, enabling the models to learn patterns and associations during the training phase. Next, feature extraction techniques are applied to capture relevant patterns and characteristics from the datasets. These features may include statistical measures, temporal dependencies, or spatial relationships that represent different aspects of the activities. The selection of appropriate features is crucial for achieving accurate classification and generalization.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

.....Error! Bookmark not defined.

CHAPTER-1.....5

INTRODUCTION.....5

 1.1 INTRODUCTION.....6

CHAPTER-2.....7

LITERATURE SURVEY.....7

 2.1 LITERATURE REVIEW.....7-8

CHAPTER-3.....9

THEORITICAL BACKGROUND.....9

 3.1 INTRODUCTION.....9

 3.2 INTRODUCTION TO PYTHON.....10-12

 3.3 BENEFITS OF PYTHON.....12-21

CHAPTER-4.....22

SYSTEM ANALYSIS.....22

 4.1 EXISTING SYSTEM:22

 4.1.1 DISADVANTAGES OF EXISTING SYSTEM:22

 4.2 PROPOSED SYSTEM:22

 4.2.1 ADVANTAGES OF PROPOSED SYSTEM:23

CHAPTER- 5.....24

SYSTEM DESIGN.....24

 5.1 INTRODUCTION.....24

 5.2 MODULES.....24

 5.2.1 DATASET:24

 5.2.2 PREPROCESSING:24

 5.2.3 GRAPHS:24

5.2.4 PREDICTION:	25-33
5.3 SYSTEM ARCHITECTURE.....	32
5.4 UML DAIGRAMS.....	34-38
5.4.1 CONSTRUCTION OF USE CASE DIAGRAMS:	38
5.4.2 SEQUENCE DIAGRAMS:.....	39
5.4.3 CLASS DIAGRAM:.....	40
5.4.4 ACTIVITY DIAGRAM:	41
5.4.5 DEPLOYMENT DIAGRAM.....	43
5.4.6 COLLABORATION DIAGRAM.....	43
CHAPTER-6.....	44
SYSTEM REQUIREMENTS.....	44
6.1 SYSTEM REQUIREMENTS.....	44
6.1.1 HARDWARE REQUIREMENTS:	44
6.1.2 SOFTWARE REQUIREMENTS:.....	44
CHAPTER-7.....	45
SYSTEM IMPLEMENTATION.....	45
7.1 INPUT AND OUTPUT DESIGNS	45
7.1.1 LOGICAL DESIGN	45
7.1.2 PHYSICAL DESIGN	46
7.2 INPUT & OUTPUT REPRESENTATION	46
7.2.1 INPUT DESIGN.....	46
7.2.2 OBJECTIVES.....	47
7.2.3 OUTPUT DESIGN	47
CHAPTER-8.....	52
SYSTEM TESTING.....	52
8.1 INTRODUCTION:.....	52

8.2 LEVELS OF TESTING	52
8.2.1 BLACK BOX TESTING.....	53
8.2.2 WHITE BOX TESTING	55
CHAPTER-9.....	62
OUTPUT SCREENS.....	62
CONCLUSION.....	65
REFERENCES.....	66

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION

The project focuses on Human Activity Recognition using machine learning techniques based on datasets. Human activity recognition involves the automatic identification and classification of human activities based on labeled data samples. This field has gained significant interest due to its wide range of applications in healthcare, sports, security, and more. Accurately recognizing and classifying human activities can have significant implications across various domains. For example, in healthcare, it can assist in understanding patients' behavior patterns and monitoring their well-being. In sports, it can provide valuable insights into athletes' performance, training effectiveness, and injury prevention strategies. In security, it can aid in identifying suspicious activities or abnormal behavior in public spaces or critical infrastructure. The project aims to develop machine learning models that can effectively recognize and classify different human activities using pre-existing datasets. These datasets contain labeled samples of various human activities, allowing the models to learn patterns and associations between features and activity classes. The project leverages a variety of machine learning algorithms, including supervised learning techniques such as decision trees, support vector machines, or deep learning architectures, to train models on the provided datasets. These algorithms learn from the labeled samples to build accurate classifiers capable of recognizing and classifying new instances of human activities. The performance of the developed models is evaluated using standard evaluation metrics such as accuracy, precision, recall, or F1 score. This evaluation helps assess the models' effectiveness and their ability to generalize well to unseen data.

By developing robust machine learning models for Human Activity Recognition, the project aims to contribute to the advancement of automated activity recognition systems. These systems can be deployed in real-world applications to enhance healthcare monitoring, sports analysis, security surveillance, and more. Overall, this project's significance lies in its potential to create reliable and efficient models for human activity recognition based on existing datasets. By leveraging machine learning techniques, it seeks to improve our understanding of human.

CHAPTER- 2

LITERATURE SURVEY

2.1 LITERATURE REVIEW

This survey paper provides a comprehensive overview of human activity recognition (HAR) methods and datasets. The authors discuss the different types of sensors that can be used for HAR, as well as the different machine learning algorithms that have been used for HAR. They also discuss the challenges and limitations of HAR, and they provide recommendations for future research[1].

This survey paper focuses on the use of deep learning for HAR. The authors discuss the different types of deep learning models that have been used for HAR, as well as the different datasets that have been used for training and evaluating these models. They also discuss the challenges and limitations of using deep learning for HAR, and they provide recommendations for future research[2].

This survey paper provides an overview of machine learning-based HAR for diverse applications. The authors discuss the different types of machine learning algorithms that have been used for HAR, as well as the different application domains that have been studied. They also discuss the challenges and limitations of using machine learning for HAR, and they provide recommendations for future research[3].

This literature survey paper provides an overview of the different methods that have been used for Human Action Recognition. The author discusses the different types of sensors that can be used for HAR, as well as the different machine learning algorithms that have been used for HAR. He also discusses the challenges and limitations of HAR, and he provides recommendations for future research[4].

This literature survey paper provides an overview of the different methods that have been used for Human Activity Recognition using machine learning. The author discusses the different types of machine learning algorithms that have been used for HAR, as well as the different datasets that have been used for training and evaluating these models. He also discusses the challenges and limitations of using machine learning for HAR, and he provides recommendations for future research[5].

This survey paper focuses on the use of vision sensors for HAR. The authors discuss the different types of vision sensors that can be used for HAR, as well as the different machine learning algorithms that have been used for HAR. They also discuss the challenges and limitations of using vision sensors for HAR, and they provide recommendations for future research[6].

CHAPTER-3

THEORETICAL BACKGROUND

3.1 INTRODUCTION:

Problem statement

The problem addressed in this project is the need for accurate and generalized human activity recognition using machine learning techniques based on labeled datasets. Challenges include limited and imbalanced datasets, extracting meaningful features, developing efficient models, handling real-time processing, and addressing privacy concerns. The goal is to overcome these challenges and develop practical activity recognition models that can be deployed in healthcare, sports, security, and other domains.

Objectives

The objective of this project is to develop a human activity recognition system using the Convolutional Neural Network (CNN) algorithm. The system will take input videos as the primary data source and aim to accurately predict and classify various human activities. The project aims to contribute to the field of Human Activity Recognition by leveraging CNN algorithms on video data, ultimately providing accurate and real-time predictions of human activities for various applications.

3.2 INTRODUCTION TO PYTHON

Python Technology

Python technology is both a programming language and a platform.

The python Programming Language

THE PYTHON PROGRAMMING LANGUAGE IS A HIGH-LEVEL LANGUAGE THAT CAN BE CHARACTERIZED BY ALL OF THE FOLLOWING BUZZWORDS:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Python programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Python byte codes —the platform-independent codes interpreted by the interpreter on the Python platform. The interpreter parses and runs each Python byte code

instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.

FEATURES OF MACHINE LEARNING

- It is nothing but automating the Automation.
- Getting computers to program themselves.
- Writing Software is bottleneck.
- Machine learning models involves machines learning from data without the help of humans or any kind of human intervention.
- Machine Learning is the science of making of making the computers learn and act like humans by feeding data and information without being explicitly programmed.
- Machine Learning is totally different from traditionally programming, here data and output is given to the computer and in return it gives us the program which provides solution to the various problems. Below is the figure.

Traditional Programming vs Machine Learning

- Machine Learning is a combination of Algorithms, Datasets, and Programs.
- There are Many Algorithms in Machine Learning through which we will provide us the exact solution in predicting the disease of the patients.
- How Does Machine Learning Works?

- Solution to the above question is Machine learning works by taking in data, finding relationships within that data and then giving the output.

Machine Learning Model

- There are various applications in which machine learning is implemented such as Web search, computing biology, finance, e-commerce, space exploration, robotics, social networks, debugging and much more.
- There are 3 types of machine learning supervised, unsupervised, and reinforcement.

BENEFITS OF PYTHON

- Presence of Third-Party Modules
- Extensive Support Libraries
- Open Source and Community Development
- Learning Ease and Support Available
- User-friendly Data Structures
- Productivity and Speed
- Highly Extensible and Easily Readable Language.

Python

Python is high level language and it is also integrated version of the program. Python is an object-oriented approach and its main aim to help programmers to write the code clearly, logical code for small and large scale of project.

Python is dynamically typed and garbage collected it also support multiple programming and it is both procedure and object oriented and also functional programming. And structural programming also supported. It has many built in function it also supports filter, map and reduce function. All the machine learning algorithm and the libraries are being supported by the python programming language. Python also support list, dict, sets and other generators. Python code can be run in different platform such as anaconda, PyCharm etc.

The main goal of this programming language is as follows:

- Python is simple, object-oriented programming language.
- The language and implementation should provide support for software engineering principles such as strong type library preset for different machine learning algorithm, and all other algorithm in simple manner.
- Coding will be smooth in python and the data analysis can be easily done in python.

This is so much so to the point where we now have modules and APIs at our disposal, and you can engage in machine learning very easily without almost any knowledge at all of how it works. With the defaults from Scikit-learn, you can get 90-95% accuracy on many tasks right out of the gate. Machine learning is a lot like a car, you do not need to know much about how it works in order to get an incredible amount of utility from it.

Despite the apparent age and maturity of machine learning, I would say there's no better time than now to learn it, since you can actually use it. Machines are quite powerful, the one you are working on can probably do most of this series quickly. Data is also very plentiful lately.

Anaconda

Anaconda is free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine Learning applications, Large- scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. It is developed and maintained by Anaconda, Inc. The distribution includes data-science packages suitable for Windows, Linux, and macOS. Packaged versions are required and are managed by the package management system anaconda. This package manager was spun out as a separate open-source package as it ended up being useful on its own and for other things than Python. There is also a small, bootstrap version of Anaconda called Miniconda, which includes only conda, Python, the packages they depend on, and a small number of other packages.

Jupyter notebook

Jupyter Notebook or so called IPython Notebook is an interactive web based computational mean for starting with Jupiter Notebook documents. The term notebook itself is a huge entity to represent the integration with different entity sets. JSON is the main document form from the same for the execution which follows the brief on the schema and the input and output means. It has high integration with several language set and has various flexibilities with the choices. The extension used for the same is “.ipynb” which runs in this platform. It’s an open-source software package with interactive communication means. It has its open standards for the same. It’s an open community best for budding programmers . The flexibility of the same is phenomenon and splendidly done the configuration and integration of the same is simplest and easy on hold so that no prior distortion is generated and the efficiency of the same is measured through out any system of choice.

It's the best software sets that been used across cross for designing and developing of the products and support wide help support. Not only to that, it provides scalability in the code and the deployment of the same. Various Language can be changed and the project can be undertaken on the same. The created notebook files can be shared and stored in various means for further utilization. It supports cultivated and interactive output sets. Easily crossed over for graphing, plotting and visualizing of the elements. Data Integration of the same is to it's best. The integration of big data and it can process chunks of values in an approx. time which gives a better performance and the higher computational means. Various works on data like cleaning, cleansing, transforming modeling and visualizing can be done by the same

Machine learning is the ability that gives the computer to learn without being explicitly programmed. There are two types of machine learning:

Supervised Learning: supervised learning is the learning of the labelled data. It is the types of machine learning that maps the input and output based on the examples input-output pairs. In supervised learning each training data having pairs of input and desired outputs values. Supervised learning algorithm analyzes the training data and produces a function which can be used for mapping of new data.

Supervised Learning The output to solve the supervised learning algorithm are as:

- Determine the types of data, before doing anything else the user should understand which types of data set is to be used for training the data.
- Gathered the training data sets either in form of human experts or from measurements.
- Determine the feature of inputs from the learned data and depends on the inputs it changed into feature vector; number of features should not be large but should contains enough information to accurately predict the outputs.

- Check the learned function and the learned algorithm for example we use support vector machines or decisions tree.
- Complete the design and run the trained data sets.
- Analyzed the output and verify the data sets to get the accurate outputs.

Unsupervised Learning:

Unsupervised learning is a type of machine learning that helps in finding the previously unknown patterns in the data set without any known labels. It is known as self- organization and allows modelling probability densities of given inputs.

unsupervised Learning Some of the algorithm used in unsupervised learning are:

- Clustering
- Anomaly detection
- Neural networks
- Approach for learning latent variable models
- Non labelled data

Semi Supervised Machine Learning algorithm: It's like the middle man which have some labeled data and some unlabeled which can be prosed by the both the structured and unsupervised learning.

The algorithms have been compared based upon the parameters: Size of the dataset and Number of technical indicators used. Accuracy and F-measure values have been computed for each algorithm. Long term model has been used to compute the accuracy and F-measure.

Reinforcement Learning: This type of learning is used to reinforce or strengthen the network based on critic information. That is, a network being trained under reinforcement learning, receives some feedback from the environment. However, the feedback is evaluative and not instructive as in the case of supervised learning. Based on this feedback, the network performs the adjustments of the weights to obtain better critic information in future.

This learning process is similar to supervised learning but we might have very less information. The following figure gives the block diagram of reinforcement learning:

```
import numpy as np
```

- NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.
- At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:
 - NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
 - The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.

- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

import time

This module provides various time-related functions. For related functionality, see also the `datetime` and `calendar` modules.

Although this module is always available, not all functions are available on all platforms. Most of the functions defined in this module call platform C library functions with the same name. It may sometimes be helpful to consult the platform documentation, because the semantics of these functions varies among platforms.

An explanation of some terminology and conventions is in order.

The epoch is the point where the time starts, and is platform dependent. For Unix, the epoch is January 1, 1970, 00:00:00 (UTC). To find out what the epoch is on a given platform, look at `time.gmtime(0)`.

The term seconds since the epoch refers to the total number of elapsed seconds since the epoch, typically excluding leap seconds. Leap seconds are excluded from this total on all POSIX-compliant platforms.

The functions in this module may not handle dates and times before the epoch or far in the future. The cut-off point in the future is determined by the C library; for 32-bit systems, it is typically in 2038.

Function `strptime()` can parse 2-digit years when given `%y` format code. When 2-digit years are parsed, they are converted according to the POSIX and ISO C standards: values 69–99 are mapped to 1969–1999, and values 0–68 are mapped to 2000–2068.

UTC is Coordinated Universal Time (formerly known as Greenwich Mean Time, or GMT). The acronym UTC is not a mistake but a compromise between English and French.

DST is Daylight Saving Time, an adjustment of the timezone by (usually) one hour during part of the year. DST rules are magic (determined by local law) and can change from year to year. The C library has a table containing the local rules (often it is read from a system file for flexibility) and is the only source of True Wisdom in this respect.

The precision of the various real-time functions may be less than suggested by the units in which their value or argument is expressed. E.g. on most Unix systems, the clock “ticks” only 50 or 100 times a second.

On the other hand, the precision of `time()` and `sleep()` is better than their Unix equivalents: times are expressed as floating point numbers, `time()` returns the most accurate time available (using Unix `gettimeofday()` where available), and `sleep()` will accept a time with a nonzero fraction (Unix `select()` is used to implement this, where available).

The time value as returned by `gmtime()`, `localtime()`, and `strptime()`, and accepted by `asctime()`, `mktime()` and `strftime()`, is a sequence of 9 integers. The return values of `gmtime()`, `localtime()`, and `strptime()` also offer attribute names for individual fields.

See `struct_time` for a description of these objects.

Changed in version 3.3: The `struct_time` type was extended to provide the `tm_gmtoff` and `tm_zone` attributes when platform supports corresponding struct tm members.

Changed in version 3.6: The `struct_time` attributes `tm_gmtoff` and `tm_zone` are now available on all platforms.

import os

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see `open()`, if you want to manipulate paths, see the `os.path` module, and if you want to read all the lines in all the files on the command line see the `fileinput` module. For creating temporary files and directories see the `tempfile` module, and for high-level file and directory handling see the `shutil` module.

Notes on the availability of these functions:

The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface; for example, the function `os.stat(path)` returns stat information about path in the same format (which happens to have originated with the POSIX interface).

Extensions peculiar to a particular operating system are also available through the `os` module, but using them is of course a threat to portability.

CHAPTER-4

SYSTEM ANALYSIS

4.1 EXISTING SYSTEM

The existing system for human activity recognition utilizes the Support Vector Machine (SVM) algorithm. SVM is a popular supervised learning technique that aims to find an optimal hyperplane to separate different classes. In the context of activity recognition, SVM is trained on labeled data to classify and recognize various human activities based on extracted features.

4.1.1 DISADVANTAGES OF EXISTING SYSTEM:

- Limited Capability to Handle High-Dimensional Data
- Difficulty in Handling Imbalanced Data
- Limited Capture of Temporal Information
- Lack of Probabilistic Outputs
- Sensitivity to Noise and Outliers
- Computational Complexity for Large Datasets

4.2 PROPOSED SYSTEM:

The proposed system utilizes the Convolutional Neural Network (CNN) algorithm for accurate human activity recognition from input video data. The system leverages CNNs' ability to capture spatial and temporal features, automate feature extraction, handle variations and noise, and incorporate temporal dependencies. It benefits from deep hierarchical learning, scalability to large datasets, and the potential for transfer learning. Overall, the proposed system offers enhanced

accuracy and performance in recognizing and classifying human activities in diverse real-world scenarios.

4.2.1 ADVANTAGES OF PROPOSED SYSTEM:

- Accurate recognition of human activities from video data.
- Automatic feature extraction without manual engineering.
- Robustness to variations and noise in the input.
- Effective capture of temporal dependencies.
- Invariance to translation and rotation in the input.
- Deep hierarchical learning for complex activity representations.
- Scalability to handle large datasets efficiently.
- Potential for transfer learning and utilization of pretrained models.

CHAPTER- 5

SYSTEM DESIGN

5.1 INTRODUCTION

System Design Introduction:

The System Design Document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces.

5.2 MODULES

5.2.1 DATA COLLECTION:

IN this project we use wound dataset collected from Kaggle which has three categories of jogging, meditation, pushups. Pixel values form images are taken as input and labels are used as output and each folder has 50 images which are used for training.

5.2.2 PRE-PROCESSING:

Pre-processing is a procedure adopted to enhance the quality of images and increase visualization. In medical imaging, image processing is a crucial phase that helps to improve the images quality. This can be one of the most critical factors in achieving good results and accuracy in next phases of proposed methodology. wound images may contain a different issue that may lead to poor and low visualization of the image. If the images are poor or of low quality, it may lead to unsatisfactory results. During preprocessing phase, we performed background elimination, elimination of non-essential blood supplies, image enhancement, and noise removal.

5.2.3 TRAIN-TEST SPLIT AND MODEL FITTING:

Now, we divide our dataset into training and testing data. Our objective for doing this split is to assess the performance of our model on unseen data and to determine how well our model has generalized on training data. This is followed by a model fitting which is an essential step in the model building process.

Model Evaluation and Predictions:

This is the final step, in which we assess how well our model has performed on testing data using certain scoring metrics, I have used 'accuracy score' to evaluate my model. First, we create a model instance, this is followed by fitting the training data on the model using a fit method and then we will use the predict method to make predictions on x_test or the testing data, these predictions will be stored in a variable called y_test_hat. For model evaluation, we will feed the y_test and y_test_hat into the accuracy_score function and store it in a variable called test_accuracy, a variable that will hold the testing accuracy of our model. We followed these steps for a variety of classification algorithm models and obtained corresponding test accuracy scores.

Algorithms

CNN Architecture, Process & Inputs

Architecture:

CNNs contain a combination of layers which transform an image into output the model can understand.

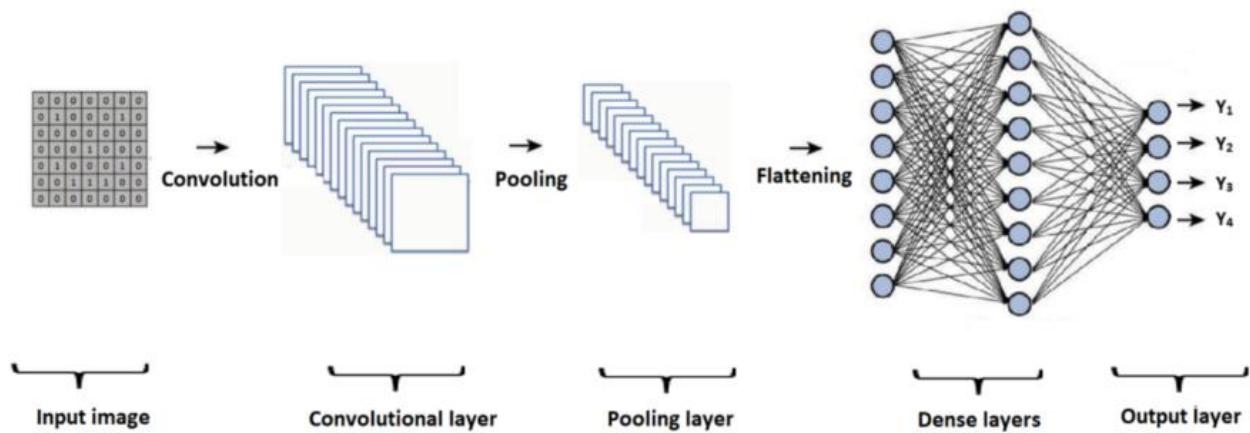


Fig 5.0 CNN Architecture

- Convolutional layer: creates a feature map by applying a filter that scans the image several pixels at a time
- Pooling layer: scales down the information generated by the convolutional layer to effectively store it
- Fully connected input layer: flattens the outputs into a single vector
- Fully connected layer: applies weights over the inputs generated by the feature analysis
- Fully connected output layer: generates final probabilities to determine the image class

Process:

Forward and backward propagation iterate through all of the training samples in the network until the optimal weights are determined and only the most powerful and predictive neurons are activated to make a prediction

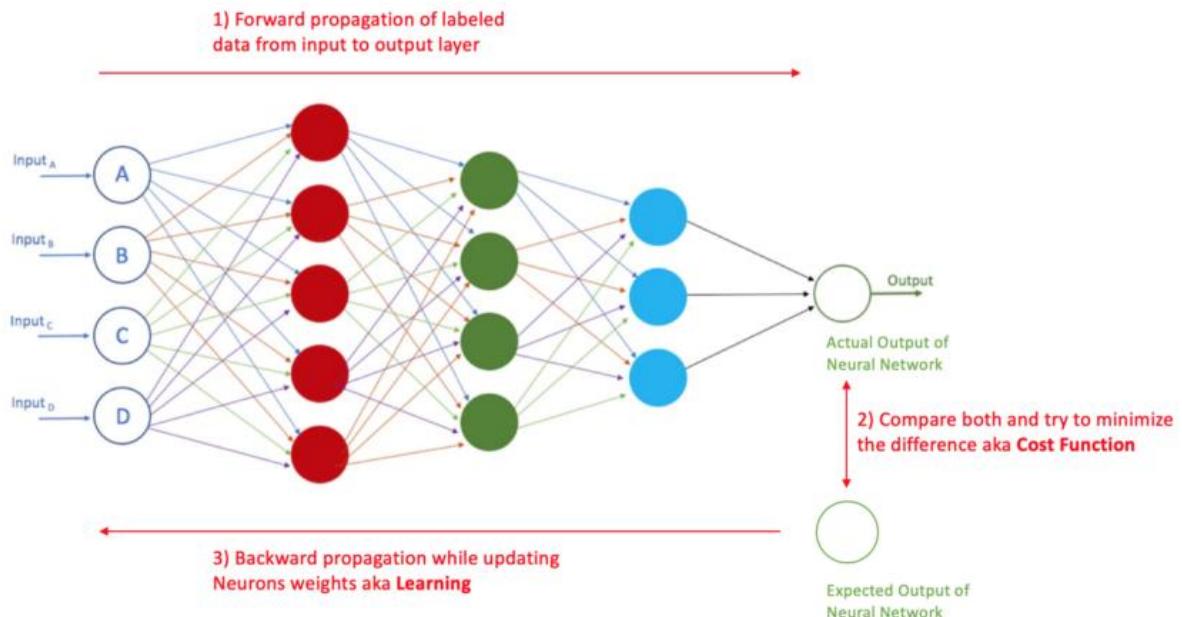


Fig 5.1

- The model trains throughout many epochs by taking one forward and one backward pass of all training samples each time
- Forward propagation calculates the loss and cost functions by comparing the difference between the actual and predicted target for each labeled image

- Backward propagation uses gradient descent to update the weights and bias for each neuron, attributing more impact on the neurons which have the most predictive power, until it arrives to an optimal activation combination
- As the model sees more examples, it learns to better predict the target causing the loss measure to decrease
- The cost function takes the average loss across all samples indicating overall performance

Inputs:

Model inputs always have to be in a 4D array consisting of (batch_size, height, width, depth)

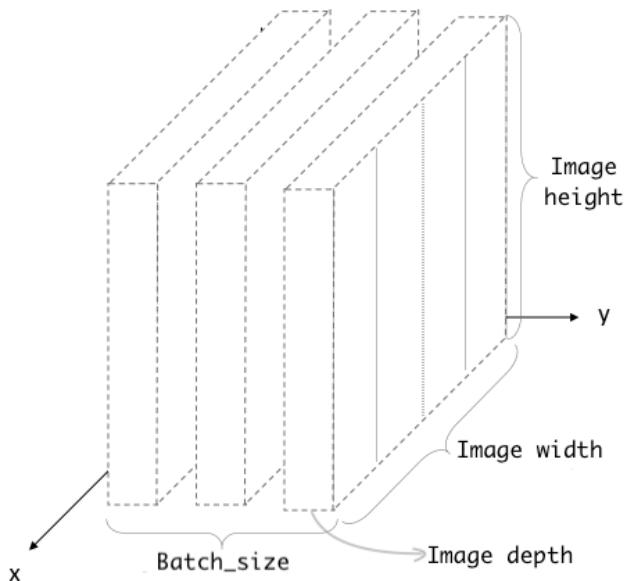


Fig 5.2

- Batch size: The number of training examples in one epoch (the higher the batch size, the more memory you'll need)
- Height & Width: Pixel dimensions of your image
- Depth: Red, Green or Blue (3), or Black & White (1)

MY VGG19 MODEL

Below is an 8 step configuration of my best performing VGG19 model. VGG19 is an advanced CNN with pre-trained layers and a great understanding of what defines an image in terms of shape, color, and structure. VGG19 is very deep and has been trained on millions of diverse images with complex classification tasks. I didn't train VGG19 any further, only froze its layers and appended a shallow 2 layer network on top of it to perform my classification task of identifying images with and without trees.

1. Load your model.

```
In [21]: # Build VGG16 structure
cnn_base = VGG19(weights='imagenet',
                  include_top=False,
                  input_shape=(228, 228, 3))
print('VGG19 Loaded')
print(cnn_base.summary())
```

2. Load your data set size. In this case, the photos designated for training, testing and validation have already been randomly shuffled into different folders and manually separated between those with (target = 1) and without trees (target = 0).

```
In [25]: # Specify dataset size
batch_size = 16
nb_train_samples = 2084
nb_validation_samples = 167
nb_test_samples = len(os.listdir('data/test/tree')) + len(os.listdir('data/test/not_tree'))
```

3. Set up a function to extract and freeze the VGG-19's initial layers which process the features and labels of images under the hood. This will allow the model to apply transfer learning wherein it can recall its pre-training from millions of photos on the web.

```
In [26]: # Build extraction function to get features and labels
def extract_features(directory, sample_amount):
    features = np.zeros(shape=(sample_amount, 7, 7, 512))
    labels = np.zeros(shape=(sample_amount))
    datagen = ImageDataGenerator(rescale=1./255)
    generator = datagen.flow_from_directory(
        directory, target_size=(228, 228),
        batch_size = batch_size,
        class_mode='binary')

    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = cnn_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i = i + 1
        if i * batch_size >= sample_amount:
            break
    return features, labels
```

4. Apply the function to your training, validation & test datasets so it extracts the features and labels from all of them.

```
In [27]: # Apply extraction function to 3 datasets
train_features, train_labels = extract_features(train_folder, nb_train_samples)
validation_features, validation_labels = extract_features(val_folder, nb_validation_samples)
test_features, test_labels = extract_features(test_folder, nb_test_samples)
```

5. Make sure data is in the right shape to reflect the dimensions of your datasets.

```
In [28]: # Shape data
reshape_y = 7 * 7 * 512
train_features = np.reshape(train_features, (nb_train_samples, reshape_y))
validation_features = np.reshape(validation_features, (nb_validation_samples, reshape_y))
test_features = np.reshape(test_features, (nb_test_samples, reshape_y))
```

6. Save extracted features and labels into a ‘bottlenecked’ folder for your final classifying layer to refer to and conclude which binary category an image belongs to.

```
In [29]: # Save features and labels
os.mkdir('data/bottlenecked')
np.save('data/bottlenecked/train_features.npy', train_features)
np.save('data/bottlenecked/train_labels.npy', train_labels)
np.save('data/bottlenecked/validation_features.npy', validation_features)
np.save('data/bottlenecked/validation_labels.npy', validation_labels)
np.save('data/bottlenecked/test_features.npy', test_features)
np.save('data/bottlenecked/test_labels.npy', test_labels)
```

7. Build the image classifier final layer on top of the VGG-10 “brain” and put it to work.

```
In [30]: # Build classifier on top of VGG19
model = Sequential()

# Add dense layers on top of VGG19
# 1
model.add(Dense(256, activation='relu', input_dim=reshape_y))
# 2
model.add(Dense(1, activation='sigmoid'))

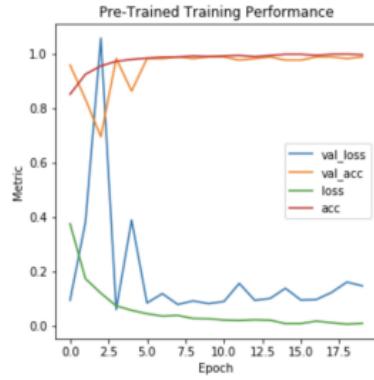
# Compile
model.compile(optimizer=optimizers.RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                     epochs=20,
                     batch_size=16,
                     validation_data=(validation_features, validation_labels))

# Save VGG19 results
model.save('models/model_VGG_01.h5')
```

8. To visualize how well your model learned using your accuracy and loss metrics, print your training history. As you can see, with each epoch (or iteration), our accuracy increase and loss decreased.

```
In [31]: # Print training history
pd.DataFrame(history.history).plot(figsize=(5, 5))
plt.title('Pre-Trained Training Performance')
plt.xlabel('Epoch')
plt.ylabel('Metric')
plt.show()
```



Beyond experimenting with different model hyper-parameters, here are some more steps you can take to improve your model:

To improve model performance, add adversity so your model can learn to recognize your target even in unlikely circumstances:

- Flip the image direction
- Incorporate images that resemble your target
- Add blurred and unsharpened versions

To improve model efficiency, reduce the model load:

- Perform max pooling in between the layers to reduce image dimensionality by compressing spatial size and parameters
- Try early stopping to prevent overfitting; if your model reaches a peak accuracy, it will stop looking for an even higher accuracy after a specific number of epochs
- Train on fewer epochs to cut the processing time
- Try a different activation function, like ReLU which only activates certain neurons, making it more efficient compared to sigmoid or tanh
- Try dropout so randomly selected neurons are ignored during training, thus creating less network computation
- Avoid large pixel images as adding more image clarity doesn't improve learning much (224 by 224 pixels is standard)

5.3 SYSTEM ARCHITECTURE

A system architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system. Organized in a way that supports reasoning about the structures and behaviors of the system. The system should function in a fraction of a second, which has unfortunately not received much attention in previous research. Although a compromise between accuracy and time is required, real-time processing is still regarded as one of the top benchmarks in information processing. Human activity recognition systems can process video frames based on frame rate per second and real-time monitoring of non-static environments, according to statistics. It remains one of the most difficult aspects of video processing to track multiple goals in a chain of online videos. This is especially true when it comes to topics such as recognizing human activity. Databases contain movements in

everyday life. These movements are considered normal, and some are considered anomalous. Because of this, recognition under dense conditions is crucial in those multiple activities. In addition, accuracy is compromised when movements overlap, such as jumping and diving together. Therefore, we plan on developing an action recognition system based on a network of video sensors in different dynamic environments. This will apply to several multispectral control videos. In order to recognize data, feature extraction and classification algorithms are required, regardless of the type of data. Support vector machines (SVM) and neural networks (NNs) can be utilized as primary classifiers in handcrafted feature extraction-oriented systems like those described in. Deep learning (DL), mainly convolutional neural networks (CNNs), based on the hierarchical system of the human visual cortex, has advanced considerably in image classification. By using feature extraction and classification models, CNNs can learn categorical information from their features. Analyzing action representations and extracting features could significantly improve action recognition.

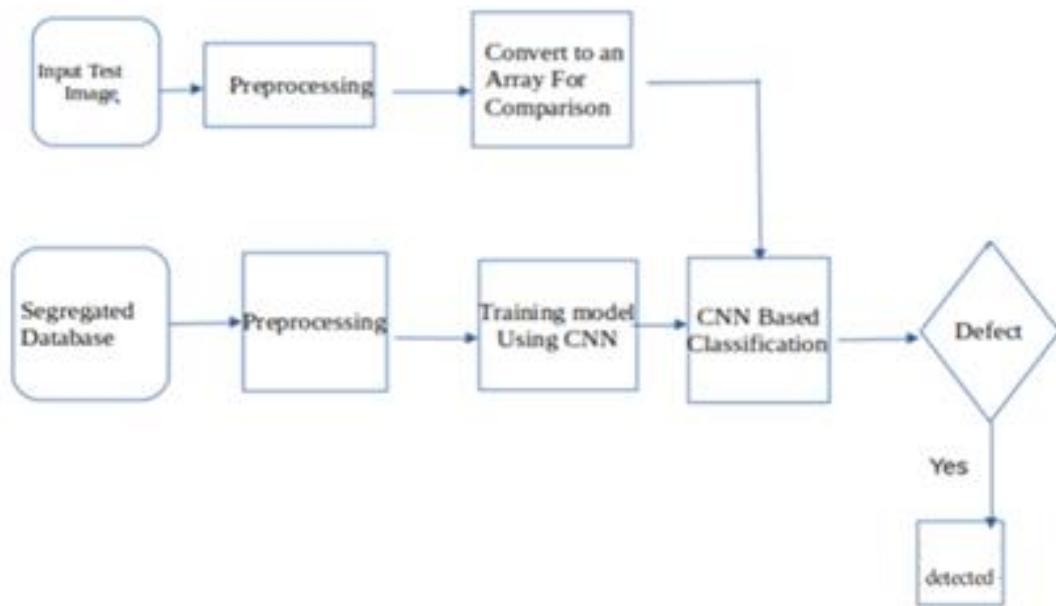


Figure 5. 3 System Architecture

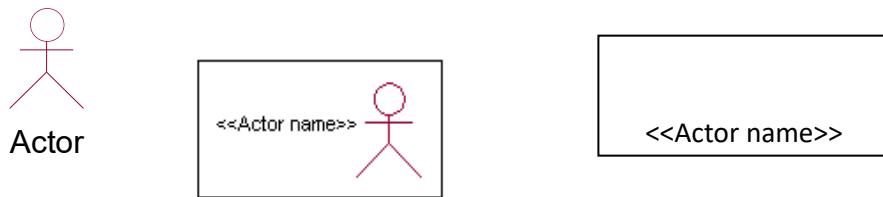
5.4 UML DAIGRAMS

Global Use Case Diagrams:

Identification of actors:

Actor: Actor represents the role a user plays with respect to the system. An actor interacts with, but has no control over the use cases.

Graphical representation:



An actor is someone or something that:

Interacts with or uses the system.

- Provides input to and receives information from the system.
- Is external to the system and has no control over the use cases.

Actors are discovered by examining:

- Who directly uses the system?
- Who is responsible for maintaining the system?
- External hardware used by the system.
- Other systems that need to interact with the system.

The actors identified in this system are:

- a. System Administrator
- b. Customer
- c. Customer Care

Identification of use cases:

Use case: A use case can be described as a specific way of using the system from a user's (actor's) perspective.

Graphical representation:



A more detailed description might characterize a use case as:

- Pattern of behavior the system exhibits
- A sequence of related transactions performed by an actor and the system
- Delivering something of value to the actor

Use cases provide a means to:

- capture system requirements
- communicate with the end users and domain experts
- test the system

Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system.

Guide lines for identifying use cases:

- For each actor, find the tasks and functions that the actor should be able to perform or that the system needs the actor to perform. The use case should represent a course of events that leads to clear goal
- Name the use cases.
- Describe the use cases briefly by applying terms with which the user is familiar.

This makes the description less ambiguous

Questions to identify use cases:

- What are the tasks of each actor?
- Will any actor create, store, change, remove or read information in the system?
- What use case will store, change, remove or read this information?
- Will any actor need to inform the system about sudden external changes?
- Does any actor need to inform about certain occurrences in the system?
- What usecases will support and maintains the system?

1.2 Flow of Events

A flow of events is a sequence of transactions (or events) performed by the system. They typically contain very detailed information, written in terms of what the system should do, not how the system accomplishes the task. Flow of events are created as separate files or documents in your favorite text editor and then attached or linked to a use case using the Files tab of a model element.

A flow of events should include:

-
- When and how the use case starts and ends

- Use case/actor interactions
- Data needed by the use case
- Normal sequence of events for the use case
- Alternate or exceptional flows

5.4.1 CONSTRUCTION OF USE CASE DIAGRAMS:

FLOW CHART:

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.

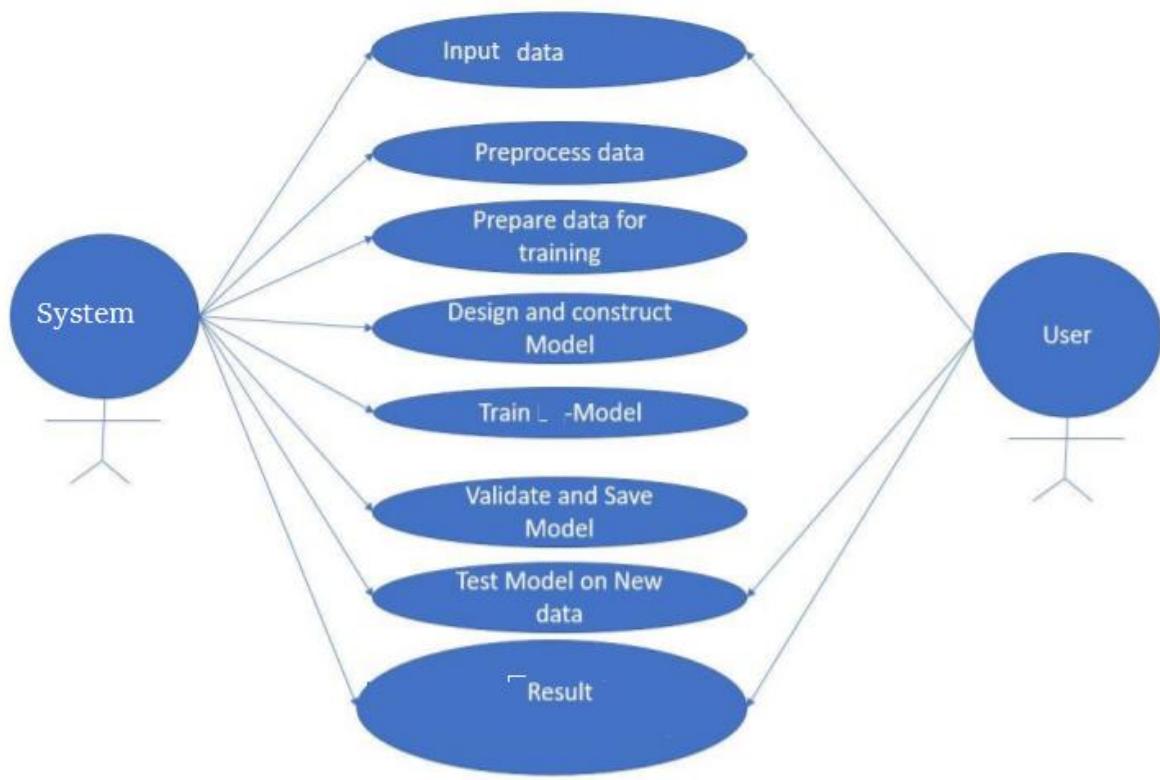


Figure 5. 4 Use Case Diagram

5.4.2 SEQUENCE DIAGRAMS:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

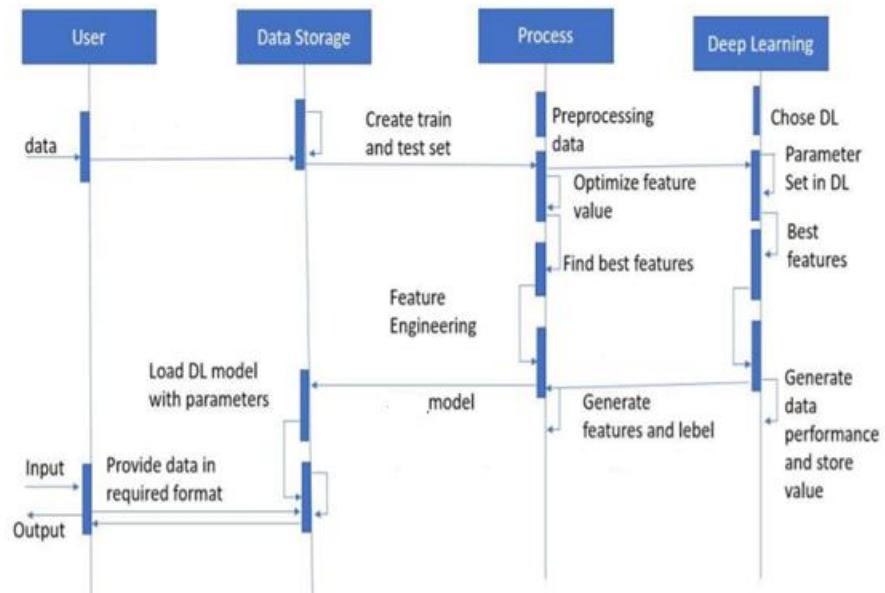


Figure 5.5 Sequence diagram

5.4.3 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

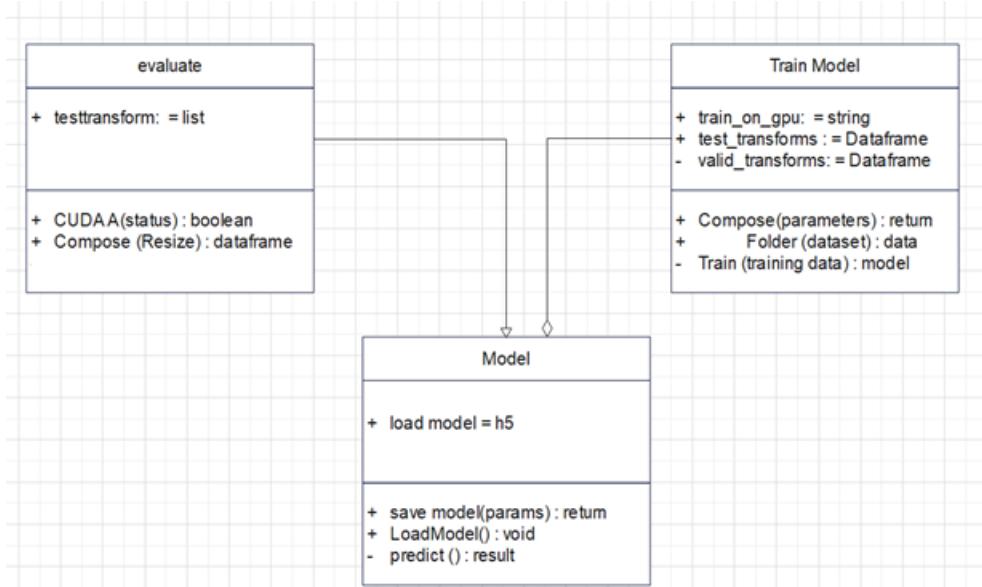


Figure 5. 6 Class Diagram

5.4.4 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.

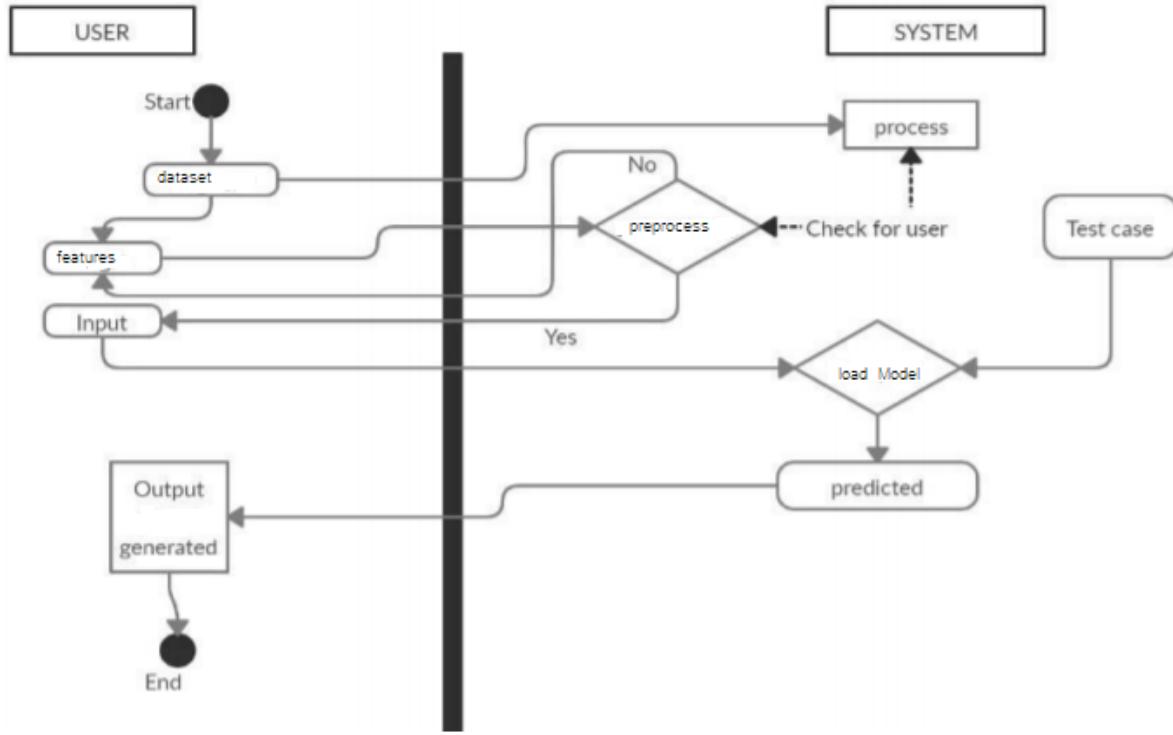


Figure 5. 7 Activity Diagram

5.4.5 DEPLOYMENT DIAGRAM

A deployment diagram in software engineering is a type of diagram that illustrates the physical deployment of software components (artifacts) to nodes (typically hardware devices or machines). It shows how software components are deployed onto hardware nodes and how those nodes are interconnected. Deployment diagrams are used to visualize the distribution of components across different nodes in a networked system, such as client-server systems or distributed systems.

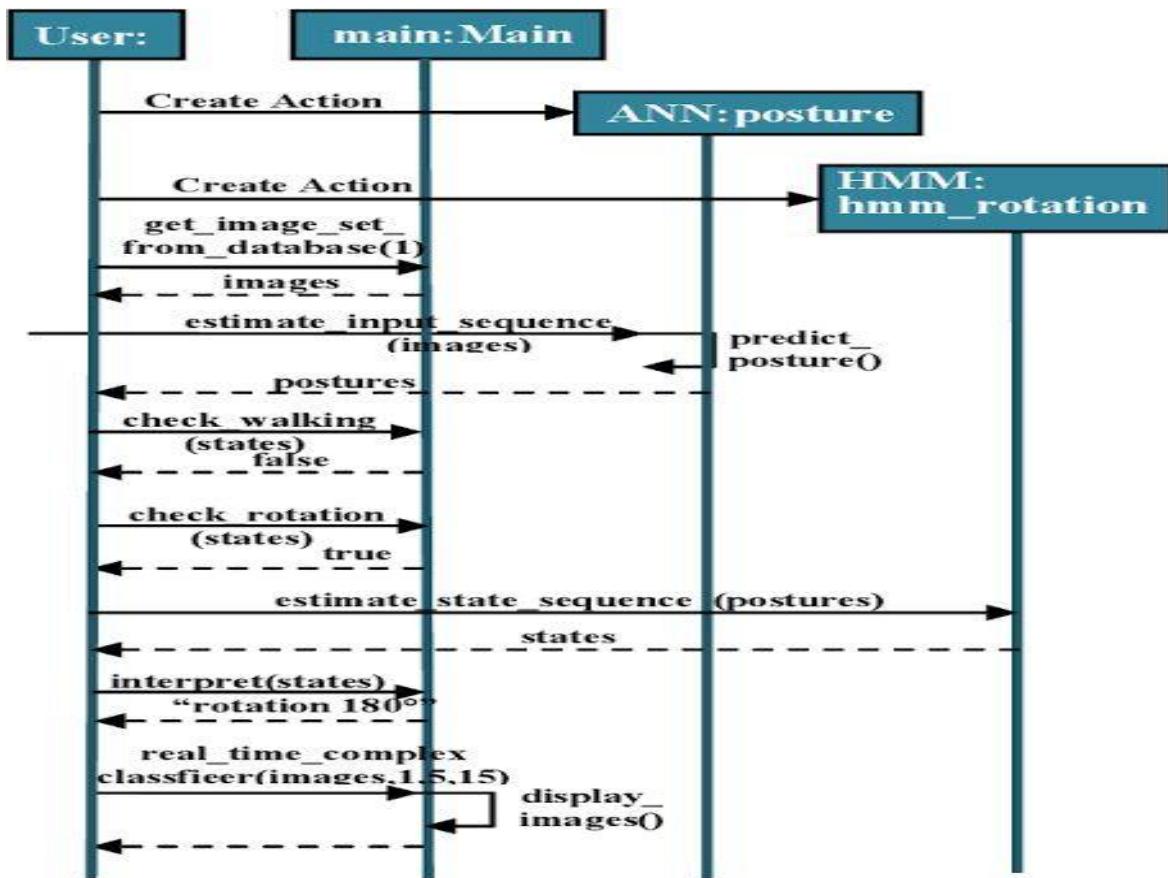


Figure 5.8 Deployment diagram

5.4.6 COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram in Unified Modeling Language (UML), is another type of interaction diagram that visualizes the interactions between objects or components within a system, focusing on the structural organization and communication paths between them.

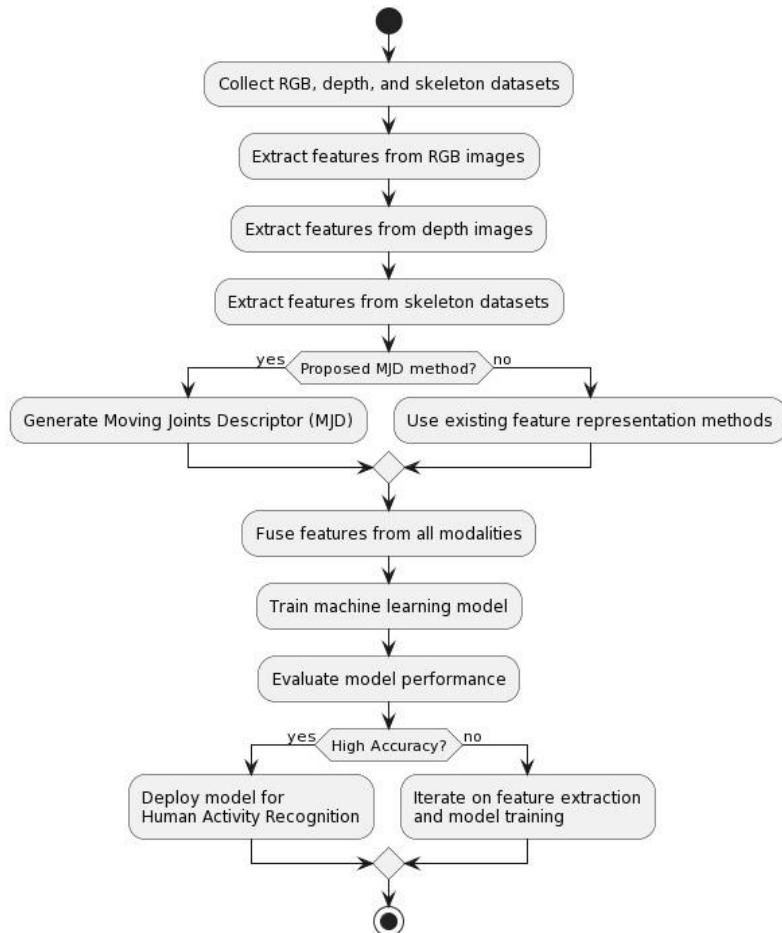


Fig 5.9 collaboration diagram

CHAPTER-6

SYSTEM REQUIREMENTS

6.1 SYSTEM REQUIREMENTS

6.1.1 HARDWARE REQUIREMENTS:

- System : Intel(R) Core(TM) i3
- Hard Disk : 1 TB.
- Ram : 4 GB.

6.1.2 SOFTWARE REQUIREMENTS:

- Operating system : Windows 7
- Coding Language : Python
- Tool : Anaconda
- Interface : OPENCV

CHAPTER-7

SYSTEM IMPLEMENTATION

To conduct studies and analyses of an operational and technological nature, and To promote the exchange and development of methods and tools for operational analysis as applied to defense problems.

7.1 INPUT AND OUTPUT DESIGNS

7.1.1 LOGICAL DESIGN

The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modeling, using an over-abstract (and sometimes graphical) model of the actual system. In the context of systems design are included. Logical design includes ER Diagrams i.e. Entity Relationship Diagrams

7.1.2 PHYSICAL DESIGN

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified / authenticated, how it is processed, and how it is displayed as output. In Physical design, following requirements about the system are decided.

1. Input requirement,
2. Output requirements,
3. Storage requirements,
4. Processing Requirements,
5. System control and backup or recovery.

Put another way, the physical portion of systems design can generally be broken down into three sub-tasks:

1. User Interface Design
2. Data Design
3. Process Design

User Interface Design is concerned with how users add information to the system and with how the system presents information back to them. Data Design is concerned with how the data is represented and stored within the system. Finally, Process Design is concerned with how data moves through the system, and with how and where it is validated, secured and/or transformed as it flows into, through and out of the system. At the end of the systems design phase, documentation describing the three sub-tasks is produced and made available for use in the next phase.

Physical design, in this context, does not refer to the tangible physical design of an information system. To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc. It involves a detailed design of a user and a product database structure processor and a control processor. The H/S personal specification is developed for the proposed system.

7.2 INPUT & OUTPUT REPRESENTATION

7.2.1 INPUT DESIGN

The input design is the link between the information system and the user. It takes the input as video and stores in the computer memory and converts the videos into frames. All those frames are then

Predicted by computer and provides the output. The accuracy of the given output is rapid and the video must be provided in seconds rather than minutes, because lesser the video length more the frames that makes computer to predict faster. If video length is more than a minute then need to create more frames that makes complex to predict.

7.2.2 OBJECTIVES

Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

1.2.3 OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. Here output is generated by using anaconda powershell prompt ,after commands are executed successfully then it provides address. That address need to be copied and paste in browser so that it opens in new tab as human activity recognition page. Then need to upload video that converts into frames and provide output after prediction. In output design it is

determined how the information is to be displayed for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

- a. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
- b. Select methods for presenting information.
- c. Create document, report, or other formats that contain information produced by the system.

Code

```
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt

IMAGE_SIZE = [224,224]

train_path = "cancer/"

from keras.preprocessing.image import ImageDataGenerator

train_datagen =
ImageDataGenerator(rescale=1./255, horizontal_flip=True, zoom_range=0.2,
validation_split=0.15)

training_set = train_datagen.flow_from_directory(
    train_path, target_size=(224,224),
batch_size=32, class_mode='categorical',
    subset='training')

validation_set = train_datagen.flow_from_directory(
    train_path, target_size=(224,224),
batch_size=32, class_mode='categorical', shuffle = True,
    subset='validation')

from tensorflow.keras.applications import VGG19
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout

## We are initialising the input shape with 3 channels rgb and weights
as imagenet and include_top as False will make to use our own custom
inputs
```

```

mv =
VGG19(input_shape=IMAGE_SIZE+[3],weights='imagenet',include_top=False)

for layers in mv.layers:
    layers.trainable = False

# if u want to add more folders and train then change number 4 to 5 or
# 6 based on folders u have to train
x = Flatten()(mv.output)
prediction = Dense(4,activation='softmax')(x)

# In[7]:


model = Model(inputs=mv.input,outputs=prediction)

model.summary()

import tensorflow as tf

class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self,epoch,logs={}):
        if(logs.get('loss')<=0.05):
            print("\nEnding training")
            self.model.stop_training = True
# initiating the myCallback function
callbacks = myCallback()

## Let us compile the model with Adam optimizer and loss function
categorical_crossentropy and metrics as categorical_accuracy
from tensorflow.keras.optimizers import Adam
model.compile(optimizer=Adam(lr=0.0001),loss='categorical_crossentropy',
'metrics=['categorical_accuracy'])

history = model.fit(training_set,
                    validation_data=validation_set,
                    epochs=50,
                    verbose=1,
                    steps_per_epoch=len(training_set),
                    validation_steps=len(validation_set),
                    callbacks = [callbacks]

```

```

        )

acc = history.history['categorical_accuracy']
val_acc = history.history['val_categorical_accuracy']

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

epochs = range(len(acc))

import matplotlib.pyplot as plt
plt.plot(epochs, acc)
plt.plot(epochs, val_acc)
plt.title("Training and validation Accuracy")
plt.savefig('accuracytrain.png')

plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
plt.title("Training and validation Loss")
plt.savefig('validationaccuracy.png')

model.save("cancer.h5")

#from tensorflow.keras.models import load_model
#from tensorflow.keras.preprocessing import image
#import numpy as np

# dimensions of our images
#img_width, img_height = 224,224

# load the model we saved
#model = load_model('content.h5')
# predicting images
#img = image.load_img('content/Train/Cars/C0.jpg',
#target_size=(img_width, img_height))
#x = image.img_to_array(img)
#x = np.expand_dims(x, axis=0)

#classes = model.predict(x)
#print (classes)

```

Chapter-8

SYSTEM TESTING

8.1 INTRODUCTION:

Testing is the debugging program is one of the most critical aspects of the computer programming triggers, without programming that works, the system would never produce an output of which it was designed. Testing is best performed when user development is asked to assist in identifying all errors and bugs. The sample data are used for testing. It is not quantity but quality of the data used the matters of testing. Testing is aimed at ensuring that the system was accurately and efficiently before live operation commands.

Testing objectives:

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say, testing is a process of executing a program with intent of finding an error.

- 1 A successful test is one that uncovers an as yet undiscovered error.
- 2 A good test case is one that has probability of finding an error, if it exists.
- 3 The test is inadequate to detect possibly present errors.
- 4 The software more or less confirms to the quality and reliable standards.

8.2 LEVELS OF TESTING

Code testing:

This examines the logic of the program. For example, the logic for updating various sample data and with the sample files and directories were tested and verified.

Specification Testing:

Executing this specification starting what the program should do and how it should be performed under various conditions. Test cases for various situation and combination of conditions in all the modules are tested.

Unit testing:

In the unit testing we test each module individually and integrate with the overall system. Unit testing focuses verification efforts on the smallest unit of software design in the module. This is also known as module testing. The module of the system is tested separately. This testing is carried out during programming stage itself. In the testing step each module is found to work satisfactorily as regard to expected output from the module. There are some validation checks for fields also. For example, the validation check is done for varying the user input given by the user which validity of the data entered. It is very easy to find error debut the system.

Each Module can be tested using the following two Strategies:

- 1 Black Box Testing
- 2 White Box Testing

8.2.1 BLACK BOX TESTING

What is Black Box Testing?

Black box testing is a software testing techniques in which functionality of the software under test (SUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on the software requirements and specifications.

In Black Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.



The above Black Box can be any software system you want to test. For example : an operating system like Windows, a website like Google ,a database like Oracle or even your own custom application. Under Black Box Testing , you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

Black box testing - Steps

Here are the generic steps followed to carry out any type of Black Box Testing.

- Initially requirements and specifications of the system are examined.
- Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
- Tester determines expected outputs for all those inputs.
- Software tester constructs test cases with the selected inputs.
- The test cases are executed.
- Software tester compares the actual outputs with the expected outputs.
- Defects if any are fixed and re-tested.

Types of Black Box Testing

There are many types of Black Box Testing but following are the prominent ones -

- Functional testing – This black box testing type is related to functional requirements of a system; it is done by software testers.
- Non-functional testing – This type of black box testing is not related to testing of a specific functionality, but non-functional requirements such as performance, scalability, usability.
- Regression testing – Regression testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

8.2.2 WHITE BOX TESTING

White Box Testing is the testing of a software solution's internal coding and infrastructure. It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability. White box testing is also known as clear, open, structural, and glass box testing.

It is one of two parts of the "box testing" approach of software testing. Its counter-part, blackbox testing, involves testing from an external or end-user type perspective. On the other hand, Whitebox testing is based on the inner workings of an application and revolves around internal testing. The term "whitebox" was used because of the see-through box concept. The clear box or whitebox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "black box testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested

WHAT DO YOU VERIFY IN WHITE BOX TESTING?

White box testing involves the testing of the software code for the following:

- Internal security holes

- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object and function on an individual basis

The testing can be done at system, integration and unit levels of software development. One of the basic goals of whitebox testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

HOW DO YOU PERFORM WHITE BOX TESTING?

To give you a simplified explanation of white box testing, we have divided it into **two basic steps**. This is what testers do when testing an application using the white box testing technique:

STEP 1) UNDERSTAND THE SOURCE CODE

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

Step 2) CREATE TEST CASES AND EXECUTE

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include manual testing, trial and error testing and the use of testing tools as we will explain further on in this article.

Unit testing:

Sl # Test Case :	UTC1
Name of Test:	Load dataset
Items being tested:	Dataset features and labels are displayed or not
Sample Input:	Dataset IMAGE file
Expected output:	All features and labels should be displayed
Actual output:	Total data is displayed
Remarks:	Pass.

Sl # Test Case :	UTC2
Name of Test:	Split data
Items being tested:	Data is divided in to train and test set
Sample Input:	Test and train size
Expected output:	Dataset is divided in to 2 parts
Actual output:	Based on given test size data is divided and stored in train and test sets
Remarks:	pass

Integration Testing:

Integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing. Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. It occurs after unit testing and before validation testing. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

Bottom-up Integration

This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.

Top-down Integration

In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations. Table 6.5 shows the test cases for integration testing and their results.

SI # Test Case :	ITC1
Name of Test:	Train Model
Item being tested:	Model fit is performed
Sample Input:	Train x and train y
Expected output:	Fit is performed
Actual output:	Training is done and accuracy is displayed

Remarks:	Pass.
----------	-------

SI # Test Case :	ITC2
Name of Test:	Accuracy calculation
Item being tested:	If accuracy of each algorithm is calculated
Sample Input:	Test x and test y
Expected output:	Accuracy of each algorithm
Actual output:	Accuracy of each model
Remarks:	Pass.

System testing:

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic. System testing is important because of the following reasons:

- ❑ System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- ❑ The application is tested thoroughly to verify that it meets the functional and technical specifications.
- ❑ The application is tested in an environment that is very close to the production environment where the application will be deployed.
- ❑ System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

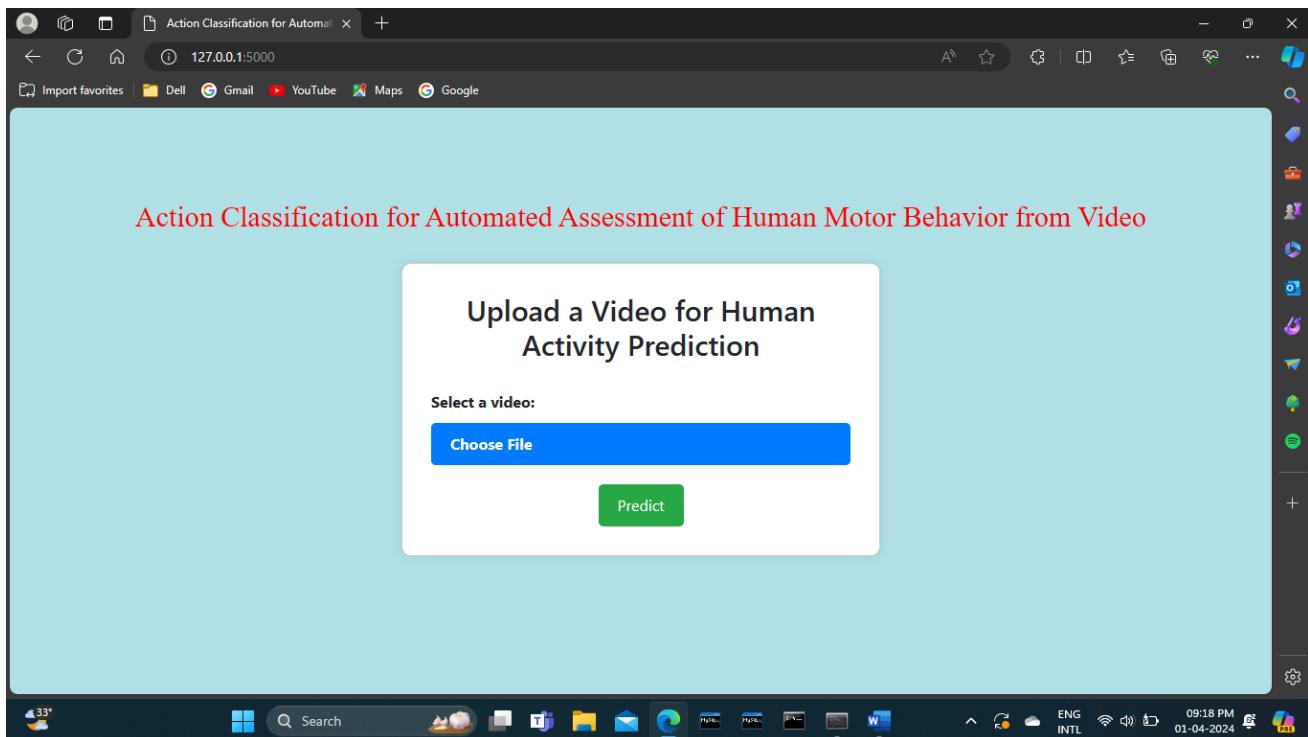
System Testing is shown in below tables.

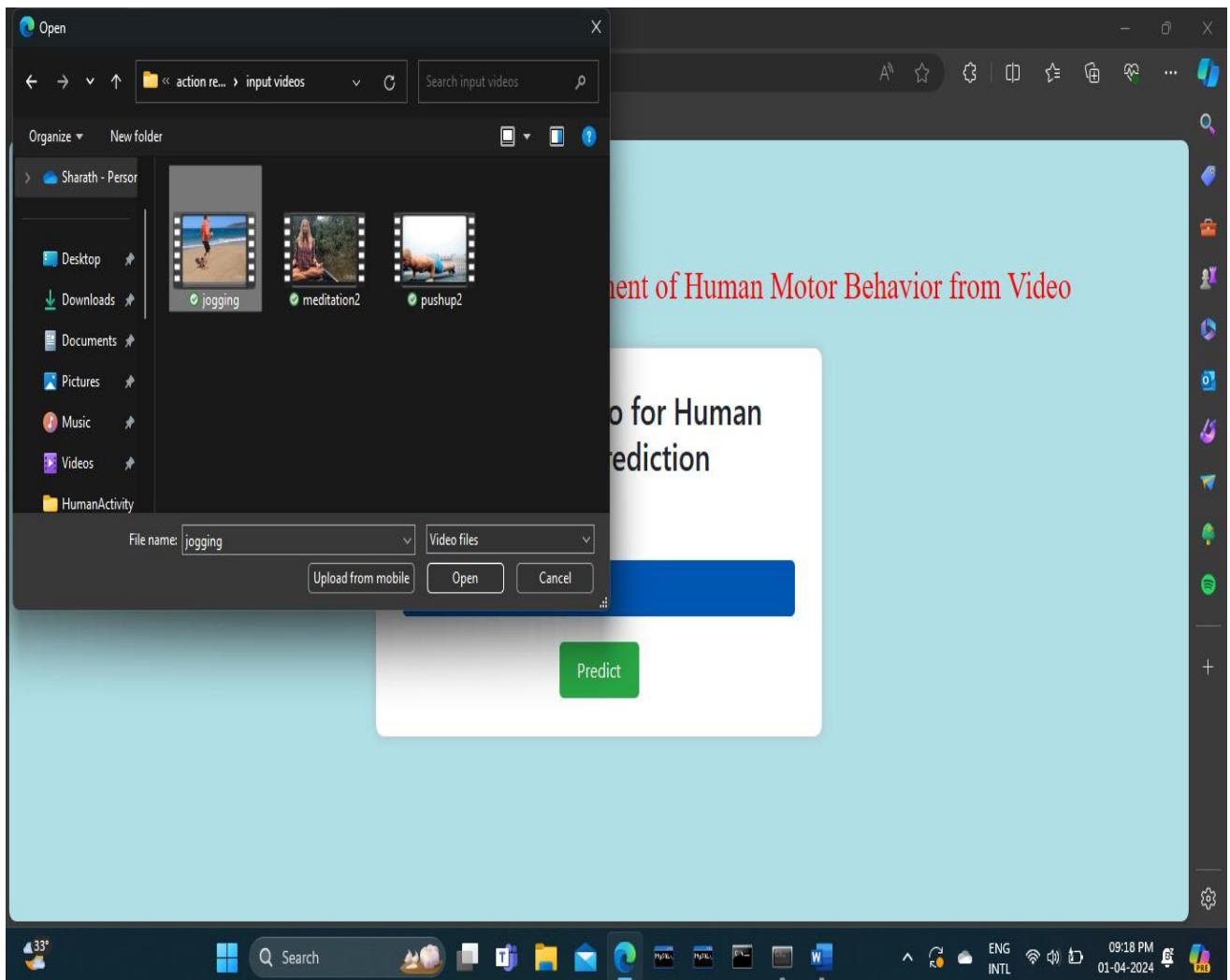
Sl # Test Case : -	STC-1
Name of Test: -	System testing in various versions of OS
Item being tested: -	OS compatibility.
Sample Input: -	Execute the program in windows XP/ Windows-7/8
Expected output: -	Performance is better in windows-7
Actual output: -	Same as expected output, performance is better in windows-7
Remarks: -	Pass

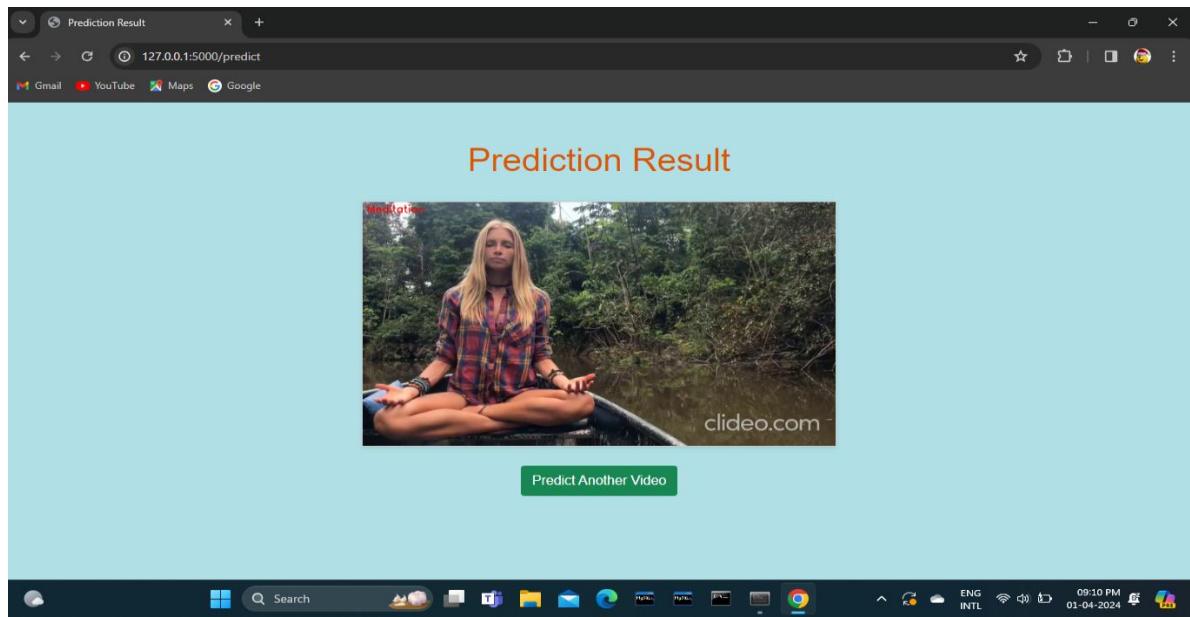
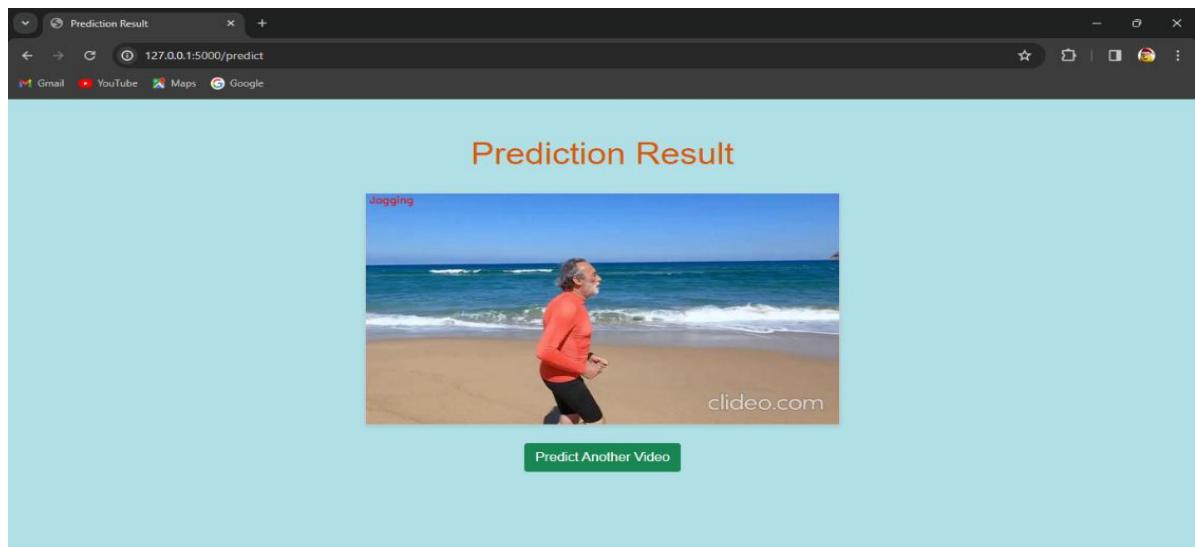
CHAPTER-9

OUTPUT SCREENS

9.1 DATASET SCREEN







CONCLUSION

The project successfully developed a Human Activity Recognition system using the CNN algorithm. The system demonstrated accurate recognition, automatic feature extraction, robustness to variations, and efficient handling of large datasets. The CNN-based system holds potential for various real-world applications and contributes to the advancement of activity recognition using machine learning techniques. Further research can build upon these findings to enhance the system and explore new applications.

Future Scope

The project holds promising future research directions for Human Activity Recognition. These include integrating multiple modalities such as audio or sensor data, exploring online and incremental learning techniques, investigating multimodal fusion approaches, enhancing explainability and interpretability, ensuring privacy preservation, optimizing real-time deployment on edge devices, leveraging transfer learning with small datasets, developing novel architectures and model compression techniques, enhancing robustness against adversarial attacks, and adapting the system to specific domains. Pursuing these avenues will contribute to advancing the accuracy, efficiency, and practicality of Human Activity Recognition in diverse applications.

REFERENCES

- [1] Yang, Y., Zhai, X., & Li, H. (2015). Human Activity Recognition: A review of state-of-the-art methods and datasets. *ACM Computing Surveys (CSUR)*, 47(3), 55.
- [2] Amini, A., Mahdian, M., & Hassani, H. (2021). A survey on deep learning for Human Activity Recognition. *Sensors*, 21(19), 6383.
- [3] Elhoseny, M., Hassanien, A. E., & Hegazy, A. (2020). A review of machine learning-based Human Activity Recognition for diverse applications. *Sensors*, 20(18), 5305.
- [4] Maskara, V. (2018). Literature survey: Human Action Recognition. arXiv preprint arXiv:1801.06552.
- [5] Tiwary, A. (2019). Human Activity Recognition using machine learning. arXiv preprint arXiv:1901.06982.
- [6] Hosseinzadeh, M., Mohammadi, A., & Najafi, H. (2022). A survey on Human Activity Recognition using vision sensors. *Sensors*, 22(8), 3242.
- [7] Wang, Y. (2016). A survey on Human Activity Recognition based on wearable sensors.
- [8] Wang, W., & Zhang, L. (2017). Human Activity Recognition based on vision: A review.
- [9] Tiwary, A., & Verma, A. (2018). A review of deep learning techniques for Human Activity Recognition.
- [10] Elhoseny, A., Hassanien, A. E., & Tolba, M. (2021). A survey on Human Activity Recognition using time series data.