# IMAGE FORGERY DETECTION

# ABSTRACT

Nowadays biometric systems are useful in recognizing person’s identity but criminals change their appearance in behaviour and psychological to deceive recognition system. To overcome from this problem we are using new technique called Deep Texture Features extraction from images and then building train machine learning model using CNN (Convolution Neural Networks) algorithm. This technique refer as LBPNet or NLBPNet as this technique heavily dependent on features extraction using LBP (Local Binary Pattern) algorithm.

In this project we are designing LBP Based machine learning Convolution Neural Network called LBPNET to detect fake face images. Here first we will extract LBP from images and then train LBP descriptor images with Convolution Neural Network to generate training model. Whenever we upload new test image then that test image will be applied on training model to detect whether test image contains fake image or non-fake image. Below we can see some details on LBP.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter No**. | | **Contents** | **Page no.** |
| 1. | 1.1 | Introduction | 1 |
|  | 1.2 | Motivation | 2 |
|  | 1.3 | Problem Definition | 2 |
|  | 1.4 | Objectives of the project | 3 |
|  | 1.5 | Scope of the project | 4 |
| 2. | 2.1 | Existing System | 5 |
|  | 2.2 | Proposed System | 9 |
|  | 2.3 | Functional Requirements | 11 |
| 3. | 3.1 | Software | 12 |
|  | 3.2 | Application Modules | 16 |
| 4. | 4.1 | Dataflow Diagram | 18 |
|  | 4.2 | Architecture Diagram | 19 |
|  | 4.3 | UML Diagrams | 20 |
| 5. | 5.1 | SDLC | 23 |
| 6. | 6.1 | Sample Code | 28 |
| 7. | 7.1 | Introduction | 35 |
|  | 7.2 | Test Cases | 40 |
| 8. | 8.1 | Input & Output Screens | 41 |
| 9. | 9.1 | Conclusion | 47 |
|  | 9.2 | Future Scope | 48 |
| 10.. | 10.1 | References | 49 |

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no** | **Fig. No** | **Figure Title** | **Page No** |
| 1 | 2.1 | Digital Watermarking | 6 |
| 2 | 2.2 | Copy-move image forgery detection | 8 |
| 3 | 2.3 | Proposed System | 10 |
| 4 | 4.1 | Dataflow Diagram | 18 |
| 5 | 4.2 | Architecture Diagram | 19 |
| 6 | 4.3 | Use Case Diagram | 20 |
| 7 | 4.4 | Class Diagram | 21 |
| 8 | 4.5 | Sequential Diagram | 22 |
| 9 | 5.1 | SDLC PHASES | 24 |
| 10 | 5.2 | Roles in SDLC | 26 |
| 11 | 5.3 | Spiral Model in SDLC | 27 |
| 12 | 7.1 | White box testing | 36 |
| 13 | 7.2 | Black box testing | 37 |
| 14 | 7.3 | Example of Unit testing | 38 |
| 15 | 7.4 | Integration testing | 39 |
| 16 | 8.1 | Home page of the application | 41 |
| 17 | 8.2 | Uploading test image | 42 |
| 18 | 8.3 | Uploading Forgery image | 42 |
| 19 | 8.4 | Set of all the forged and Original images | 43 |
| 20 | 8.5 | Uploading image 1 | 44 |
| 21 | 8.6 | Click on classify button | 44 |
| 22 | 8.7 | Classified Forgery image | 45 |
| 23 | 8.8 | Classified Original image | 46 |

|  |  |  |
| --- | --- | --- |
| **S. No** | **Table Name** | **Page No** |
| **1** | 7.2 Test Cases | 36 |

**CHAPTER - 1 INTRODUCTION**

## INTRODUCTION

In today's digital age, the proliferation of image editing tools and techniques has made creating deceptive and manipulated images easier than ever. From social media platforms to professional photography, the authenticity of images can be questioned, posing serious challenges in various domains such as journalism, law enforcement, and digital forensics. To address this issue, the development of robust image forgery detection techniques has become increasingly crucial.

This project focuses on detecting fake face images, a particularly challenging task given the prevalence of facial manipulation in various contexts, including social media profile pictures, identification documents, and surveillance footage. Traditional methods of forgery detection often rely on handcrafted features or statistical analyses, which may lack the ability to capture subtle and complex patterns indicative of forgery.

The Local Binary Pattern (LBP) operator is a powerful texture descriptor that characterizes the local patterns of pixel intensities within an image. By encoding texture information at different spatial scales, LBP offers a robust representation of image textures, making it suitable for identifying subtle alterations introduced during image manipulation processes.

To overcome these limitations, we propose the use of Local Binary Patterns (LBP) as a feature extraction method in conjunction with Convolutional Neural Networks (CNNs) for image forgery detection. LBP is a powerful texture descriptor capable of capturing local patterns within an image, making it well-suited for detecting tampering or manipulation. By integrating LBP-based features with the discriminative power of CNNs, we aim to develop a robust and efficient forgery detection system tailored specifically for fake face images.

Overall, this thesis endeavors to contribute to the ongoing efforts to combat the proliferation of fake images and preserve the integrity of visual content in the digital era.

## MOTIVATION

The motivation behind this project stems from the growing concern over the proliferation of fake or forged images, particularly fake face images, in digital media. Several factors contribute to this motivation:

* **Rise of Digital Manipulation:** With the advancement of image editing software and digital manipulation techniques, it has become increasingly easy to create fake images that appear authentic. This poses serious challenges in verifying the authenticity of visual content, especially in fields where image integrity is critical, such as journalism, law enforcement, and biometrics.
* **Impact on Trust and Credibility:** The spread of fake images can undermine trust and credibility in digital media, leading to misinformation, confusion, and potential harm to individuals and organizations. Detecting and mitigating the effects of image forgery is essential for maintaining the integrity of digital content and preserving trust in visual information sources.
* **Emerging Threats:** Fake face images, in particular, present significant threats, including identity theft, impersonation, and privacy violations. These images can be used maliciously to deceive individuals, manipulate public opinion, or perpetrate fraud. Detecting and preventing the spread of fake face images is crucial for safeguarding individuals' privacy and security in the digital age.

## PROBLEM DEFINITION

The project aims to develop an efficient and accurate system for detecting fake face images using a combination of Local Binary Patterns (LBP) and Convolutional Neural Networks (CNNs). Specifically, the project addresses the challenge of identifying manipulated regions in face images generated through forgery techniques such as face swapping, retouching, and other forms of digital manipulation.

### Key components:

* Forgery Detection: The primary objective is to distinguish between authentic and manipulated face images by detecting the presence of forgery or tampering.
* Feature Extraction: Utilizing Local Binary Patterns (LBP) to extract texture features that capture the underlying patterns and structures within the image, which are indicative of potential manipulation.
* Deep Learning Model: Developing a Convolutional Neural Network (CNN) architecture, termed LBPNET, to effectively learn discriminative features from the extracted LBP representations and classify images as authentic or fake.
* Evaluation Metrics: Assessing the performance of the proposed system using metrics such as accuracy, precision, recall, and F1 score on a test dataset to quantify its ability to correctly identify fake face images while minimizing false positives and false negatives.

## OBJECTIVES OF THE PROJECT

The main objectives of the project "Image Forgery Detection using LBP Based Machine Learning Convolution Neural Network (LBPNET) for Fake Face Images" are:

* **Developing an Efficient Detection System:** Create a robust system capable of accurately distinguishing between authentic and manipulated face images.
* **Utilizing Local Binary Patterns (LBP):** Leverage LBP feature extraction to capture texture information from images, enabling the detection of potential forgery patterns.
* **Implementing Convolutional Neural Networks (CNNs):** Design and train a CNN architecture (LBPNET) to effectively learn and classify features extracted from LBP representations, enhancing forgery detection accuracy.
* **Evaluating Performance:** Assess the performance of the proposed system using metrics such as accuracy, precision, recall, and F1 score to ensure reliable forgery detection while minimizing false positives and false negatives.
* **Enhancing Trust and Security:** Contribute to the advancement of forgery detection technologies, particularly in combating fake face images, to enhance trust, security, and integrity in digital media and online platforms.

## SCOPE OF THE PROJECT

* Focuses specifically on detecting fake face images, which are often used for malicious purposes such as identity theft or spreading misinformation.
* Utilizes a combination of LBP feature extraction and CNN-based deep learning techniques for forgery detection.
* Targets a diverse dataset encompassing both real and fake face images, covering a range of forgery techniques and variations in lighting, pose, and facial expressions.
* Aims to develop a scalable and efficient solution that can be applied to real-world scenarios, including social media platforms, digital forensics, and biometric authentication systems.

# CHAPTER - 2 SYSTEM ANALYSIS

## EXISTING SYSTEMS

### Digital Watermarking:

Digital watermarking is a technique used to embed a unique identifier or message into an image, audio, or video file. This identifier can be used to verify the authenticity and integrity of the content. In the context of image forgery detection, digital watermarking can be used to detect tampering or modifications made to an image.

There are two main approaches to digital watermarking-based image forgery detection: active and passive. Active watermarking involves embedding a watermark into the image before it is distributed, while passive watermarking does not require any prior knowledge of the image.

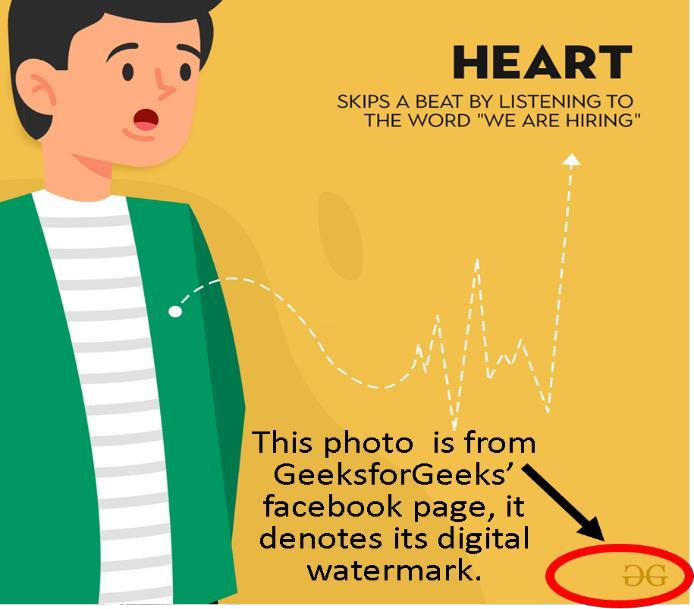
In the passive approach, any pre-embedded information, such as a watermark embedded for the detection of image forgery, cannot be relied upon. This approach is also known as the blind approach because there is no additional information for image forgery detection. The image features extracted during the acquisition phase are themselves proof of authenticity of the image.

Digital watermarking has been used widely, whether in its visible or blind form. Visible watermarking involves embedding a watermark that is visible to the human eye, while blind watermarking does not require any human intervention to detect the watermark.

Some common techniques used in digital watermarking-based image forgery detection include:

* Image artifacts caused by various irregularities as markers to determine image validity
* Color filter arrays
* Resampling, image retouching, image copy-paste, and splicing detection

In conclusion, digital watermarking-based image forgery detection is a powerful technique used to detect and prevent image tampering. It has many applications in various fields, including copyright protection, image authentication, and forensic analysis.



### Fig 2.1: Digital Watermarking

1. **Signature Based image forgery detection:**

Signature-based methods for image forgery detection rely on identifying unique characteristics or "signatures" present in images that can indicate tampering or manipulation. These signatures often manifest as inconsistencies in image properties, such as noise patterns, compression artifacts, or sensor characteristics. Some of the signature-based methods are:

* Noise Analysis: Every imaging sensor introduces some level of noise into captured images. Signature-based methods analyze noise patterns within an image to detect discrepancies that may indicate tampering. For example, if a portion of an image has been pasted from another source, it may exhibit different noise characteristics compared to the surrounding areas.
* Compression Artifacts: Lossy compression algorithms, such as JPEG, introduce artifacts into images that can be analyzed to detect tampering. Signature-based methods examine compression artifacts, such as blockiness or ringing effects, to identify regions that have been altered or manipulated.
* Sensor Pattern Noise: Imaging sensors have unique patterns of noise, known as sensor pattern noise (SPN), caused by imperfections in the sensor's manufacturing process.
* Signature-based methods utilize SPN analysis to identify regions within an image that exhibit anomalous noise patterns, suggesting potential tampering.
* Blur Analysis: Tampering with an image often involves blending or smoothing edges to conceal the boundaries between manipulated and original regions. Signature-based methods analyze blur characteristics to detect inconsistencies in sharpness or smoothness across different parts of the image.

Signature-based methods often require careful analysis and feature extraction techniques to effectively identify tampered regions within images. While they can be highly effective in certain scenarios, they may struggle with complex manipulations or cases where the forger has taken steps to conceal their tracks. As such, signature-based methods are often used in conjunction with other forgery detection techniques to provide comprehensive coverage and robustness.

### Copy–move image forgery detection:

Copy-move forgery is a type of image tampering where a portion of an image is copied and pasted onto another part of the same image. This technique is often used to duplicate objects, remove unwanted elements, or conceal information within an image.

Copy-move forgery detection is a crucial aspect of digital image authentication, which involves identifying and localizing areas in an image that have been copied and pasted from another region within the same image. This type of forgery is particularly common and can be challenging to detect, as it can be difficult to distinguish between a genuine image and a tampered one.

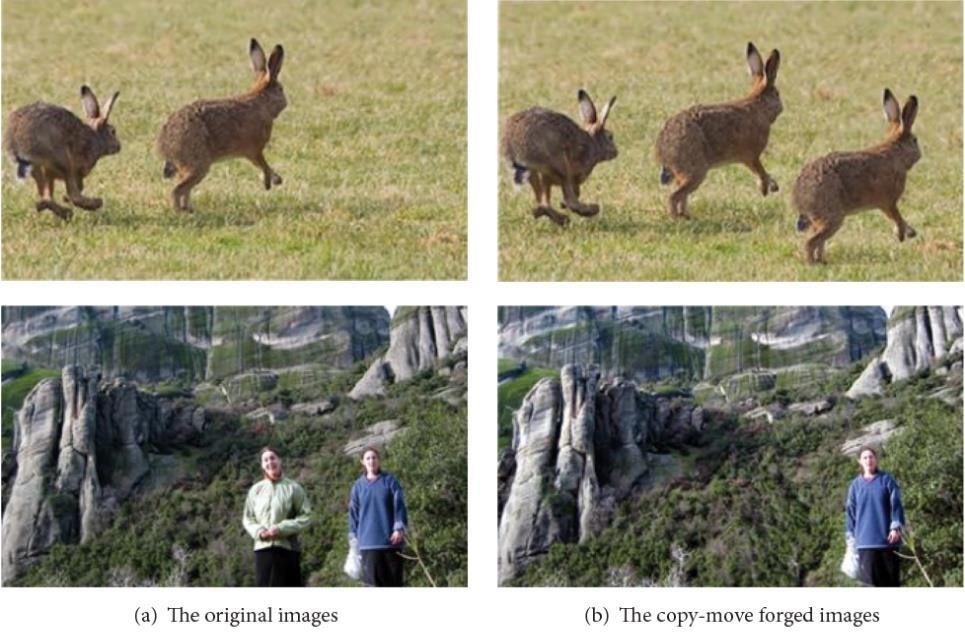
There are various approaches to copy-move forgery detection, including block-based, keypoint- based, and hybrid methods. Block-based methods divide the image into small blocks and analyze the similarity between them to detect forgery. Keypoint-based methods, on the other hand, use features such as SIFT or SURF to detect and match keypoints between the original and tampered images.

Some common techniques used in copy-move forgery detection include:

* Block-based methods: These methods divide the image into small blocks and analyze the similarity between them to detect forgery. The blocks can be represented using features such as DCT, Fourier transform, or Tetrolet transform.
* Keypoint-based methods: These methods use features such as SIFT or SURF to detect and match keypoints between the original and tampered images.
* Hybrid methods: These methods combine block-based and keypoint-based approaches to improve detection accuracy.
* Deep learning-based methods: These methods use deep neural networks to learn features and detect forgery.

Some of the challenges in copy-move forgery detection include:

* Robustness to rotation and resizing: The detection method should be able to handle images that have been rotated or resized.
* Robustness to noise and compression: The detection method should be able to handle images that have been corrupted by noise or compression.
* Handling of complex forgery: The detection method should be able to handle complex forgery attacks, such as those that involve multiple copies and pastes.



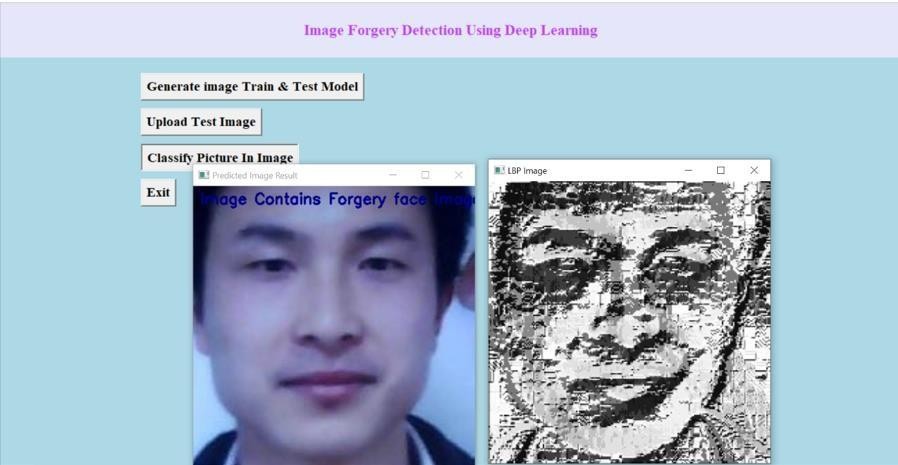
**Fig 2.2: Copy-move image forgery detection**

## PROPOSED SYSTEM

The Proposed system is to develop a robust and accurate system for detecting fake face images, which are increasingly used for malicious purposes such as identity theft, fraud, and spreading misinformation. By leveraging the complementary strengths of LBP for texture feature extraction and CNNs for deep learning-based classification, we seek to create a reliable forgery detection system specifically tailored for fake face images.

Some of the key characteristics are:

* **Focus on Fake Face Images:** The project specifically targets the detection of fake face images, which are increasingly prevalent due to advancements in face manipulation techniques. By focusing on this specific domain, the project addresses a critical need for detecting forged face images used for identity theft, fraud, and misinformation.
* **Integration of LBP and CNNs:** The project leverages the complementary strengths of Local Binary Patterns (LBP) and Convolutional Neural Networks (CNNs) to enhance forgery detection accuracy. LBP is utilized for texture feature extraction, capturing subtle patterns indicative of manipulation, while CNNs are employed for learning discriminative features and classifying images as authentic or fake.
* **Feature-Rich Representation:** By extracting texture features using LBP, the proposed approach provides a feature-rich representation of the input images, enabling the detection of subtle forgery patterns that may be overlooked by traditional methods. This enhances the robustness and effectiveness of the forgery detection system.
* **End-to-End System:** The project encompasses the entire pipeline of forgery detection, from data collection and preprocessing to model training and evaluation. This end-to-end approach ensures that all components of the system are optimized and integrated to achieve maximum detection performance.
* **Scalability and Generalization:** The proposed system aims to be scalable and generalizable, capable of detecting a wide range of forgery techniques and variations in face images. By training on diverse datasets and employing deep learning techniques, the system can adapt to new challenges and emerging forgery methods, enhancing its applicability in real-world scenarios.
* **Performance Evaluation:** The project includes thorough performance evaluation using standard metrics such as accuracy, precision, recall, and F1 score. This ensures that the proposed system's effectiveness is rigorously assessed, providing confidence in its ability to accurately detect fake face images while minimizing false positives and false negatives.



**Fig 2.3: Proposed System**

## FUNCTIONAL REQUIREMENTS

### HARDWARE:

* Processor : Intel i5
* RAM : 8 GB
* Storage : Minimum of 50 GB disk space
* Network : Internet-enabled

### SOFTWARE:

* Programming languages : Python
* Libraries Used : OpenCV, pandas, pillow, scikit and others
* Front – end : HTML, CSS
* Database Used : MySQL
* Operating System : Windows 10

# CHAPTER – 3 SOFTWARE ENVIRONMENT

## SOFTWARE

A software development environment (SDE) is the collection of hardware and software tools a system developer uses to build software systems.

There 4 different environments in a software development team are shown below:

### Development environment:

The development environment is the first environment in software development which acts as the workspace for developers to do programming and other operations related to the creation of software and/or systems.

An integrated development environment (IDE) — a software package with extensive functions for authoring, building, testing, and debugging a program which is commonly used by software developers. Some programming software tools such as Microsoft Visual Studio, Eclipse, NetBeans, and other integrated development environments.

### Testing environment:

The test environment is where testing teams evaluate the application/quality. program’s This also allows computer programmers to find out and solve any defects that may interfere with the application’s smooth operation or degrade the user experience.

The test environment is created by allocating storage, computing, and other resources needed for testing. This could include new physical/virtual devices set up for testing use cases defined by developers. For example, Selenium tests cannot run for the whole set of browsers through which you want your application to be accessible at the same time. This means that you either run tests sequentially or generate multiple test environments.

### Staging environment:

When you generate the staging instance of an application, you are confident sufficient to reveal it to the immediate owner but not to users. You should run more tests before exposing them to the latter group. The staging environment is similar to the pre-production in use.

The staging environment is frequently restricted to a small group of people. The only groups that can access the application in staging are those with whitelisted emails and IP addresses, as well as your developer team. The goal of a staging environment is to simulate production as much as possible.

### Production environment:

When the end-user uses a web/mobile application, the program is operating on a production server. It’s been created in the production environment.

Tests can be carried out while the product is in production, and new features can be introduced safely simultaneously. Feature flags allow you to show a future version of an app to a select few users while the rest continue to utilize the current version.

### Python Programming Language:

Python programming language plays a central role in various aspects of the project Image Forgery Detection using LBP Based Machine Learning Convolution Neural Network (LBPNET) for Fake Face Images.

1. **Implementation of Algorithms:** Python serves as the primary language for implementing the forgery detection algorithms. This includes writing code for feature extraction using Local Binary Patterns (LBP), designing and training Convolutional Neural Networks (CNNs), and implementing forgery detection logic.
2. **Data Preprocessing:** Python is used for data preprocessing tasks, such as loading image datasets, resizing images to a consistent size, normalizing pixel values, and augmenting the data for training purposes. Libraries like OpenCV and scikit-image are commonly employed for these tasks.
3. **Model Training and Evaluation:** Python is used to train the forgery detection model (LBPNET) using deep learning frameworks like TensorFlow or PyTorch. This involves defining the network architecture, specifying loss functions and optimization algorithms, and iterating over the dataset to update model parameters. Python is also used to evaluate the trained model's performance on test datasets, calculating metrics such as accuracy, precision, recall, and F1 score.
4. **Experimentation and Prototyping:** Python's flexibility and ease of use make it ideal for rapid prototyping and experimentation. Researchers and developers can quickly iterate on different ideas, algorithms, and model architectures within Python's interactive environment, such as Jupyter Notebooks.
5. **Visualization and Analysis:** Python libraries like Matplotlib and Seaborn are used for visualizing data, model performance, and experimental results. These libraries enable the creation of plots, charts, and graphs to gain insights into the forgery detection process, analyze model behavior, and communicate findings effectively.
6. **Integration with External Tools and Libraries:** Python's extensive ecosystem of libraries and tools facilitates integration with external software and services. For example, Python scripts can be used to interface with image editing software for generating fake face images, accessing cloud services for data storage and processing, or deploying the trained model as a web service or application.

### Deep Learning Frameworks:

Deep learning frameworks are essential tools for building, training, and evaluating the forgery detection model. TensorFlow and PyTorch are the two main deep learning frameworks commonly used:

### TensorFlow:

TensorFlow is an open-source deep learning framework developed by Google Brain. It offers a comprehensive ecosystem of tools and libraries for building and deploying machine learning models, including Convolutional Neural Networks (CNNs).

### Key Features:

* High-level APIs (such as Keras) for building neural networks with ease.
* Efficient computation using dataflow graphs and automatic differentiation.
* Scalability and portability across a range of hardware platforms, including CPUs, GPUs, and TPUs (Tensor Processing Units).
* Built-in support for distributed training and deployment in production environments.

### PyTorch:

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab. It is known for its dynamic computation graph and imperative programming paradigm, which offer flexibility and ease of use.

### Key Features:

* Dynamic computation graph: PyTorch allows for dynamic graph creation and modification during runtime, enabling more intuitive model development and debugging.
* Pythonic API: PyTorch's API is designed to be intuitive and Pythonic, making it easy to prototype, experiment, and debug deep learning models.
* Strong research support: PyTorch is widely used in academia and research due to its flexibility and ease of experimentation. It offers a rich ecosystem of libraries and tools for various deep-learning tasks.
* Growing adoption in the industry: PyTorch's ease of use and flexibility have led to its increasing adoption in the industry for developing production-ready machine learning applications.

## APPLICATION MODULES

The application modules involved in the project "Image Forgery Detection using LBP Based Machine Learning Convolution Neural Network (LBPNET) for Fake Face Images" can be organized into several key components, each serving a specific purpose within the forgery detection system. Here are the main application modules:

### Data Collection Module:

This module is responsible for collecting a diverse dataset of both real and fake face images. It may involve sourcing publicly available datasets, generating synthetic fake face images using forgery techniques, or collecting images from online sources.

### Data Preprocessing Module:

The data preprocessing module handles tasks such as resizing, normalizing, and augmenting the image dataset. It ensures that the input images are standardized and prepared for feature extraction and model training.

### Feature Extraction Module:

This module extracts texture features from the preprocessed images using Local Binary Patterns (LBP). LBP captures the local texture patterns within the images, providing a feature-rich representation for forgery detection.

### Model Training Module:

The model training module involves training the Convolutional Neural Network (CNN) architecture, LBPNET, using the extracted LBP features. It utilizes labeled training data to optimize the network parameters and learn discriminative features for classifying authentic and fake face images.

### Model Evaluation Module:

The model evaluation module assesses the performance of the trained LBPNET model using various metrics such as accuracy, precision, recall, and F1 score. It involves testing the model on a separate validation or test dataset to evaluate its ability to accurately detect fake face images.

### Visualization Module:

The visualization module creates visualizations and plots to analyze the performance of the forgery detection system. It may include visualizing training/validation curves, confusion matrices, and ROC curves to gain insights into the model's behavior and performance.

### Deployment Module:

The deployment module is responsible for deploying the trained forgery detection system into production or integrating it into existing applications. It may involve packaging the model into a deployable format, creating APIs for inference, or integrating the system into web or mobile applications.

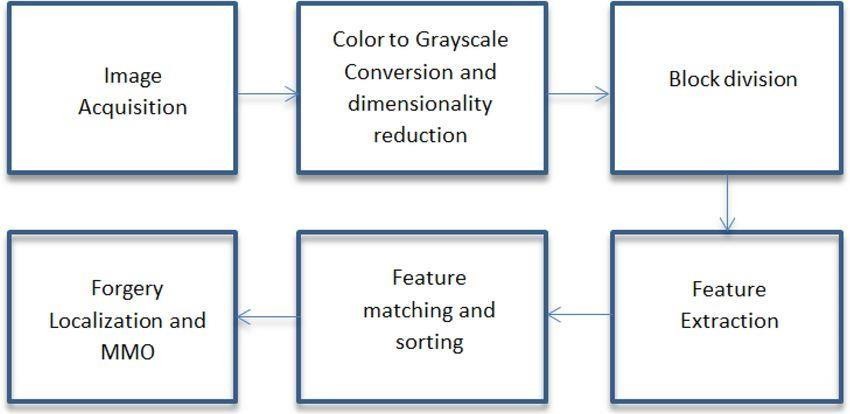
### User Interface (UI) Module:

The UI module provides a user-friendly interface for interacting with the forgery detection system. It may include features such as uploading images for analysis, displaying detection results, and configuring system parameters.

# CHAPTER – 4

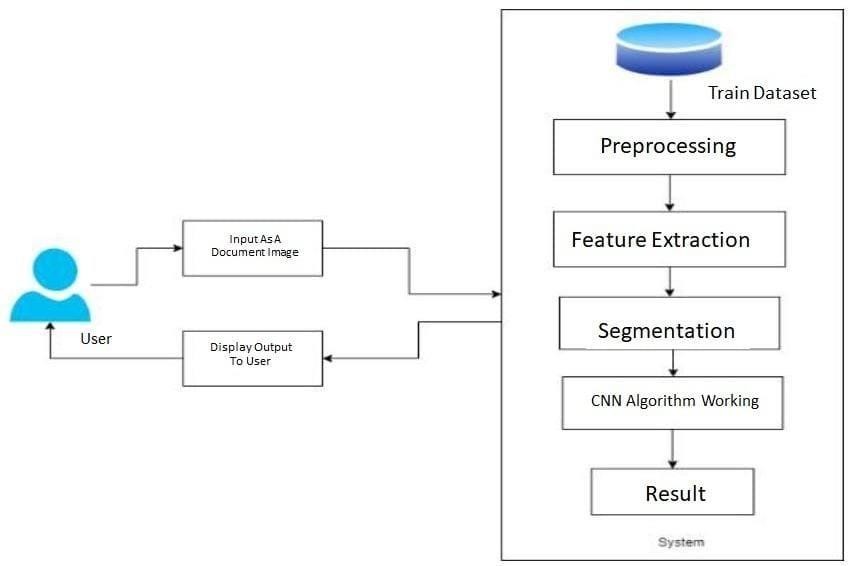
**SYSTEM DESIGN AND UML DIAGRAMS**

## DATA FLOW DIAGRAM



**Fig 4.1: Data Flow Diagram**

## SYSTEM ARCHITECTURE



**Fig 4.2: System architecture**

## UML DIAGRAMS

### Use Case Diagram:

Use case diagrams identify the functionalities provided by the use cases, the actors who interact with the system, and the association between the actors and the functionalities. The Use Cases in this system are:

Generate Image Train and Test

Exit

Model

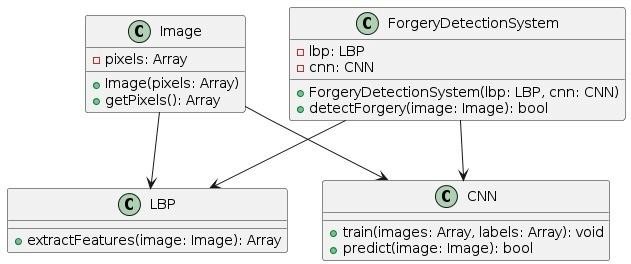
Upload Test Image

User

Classify Piture In Image

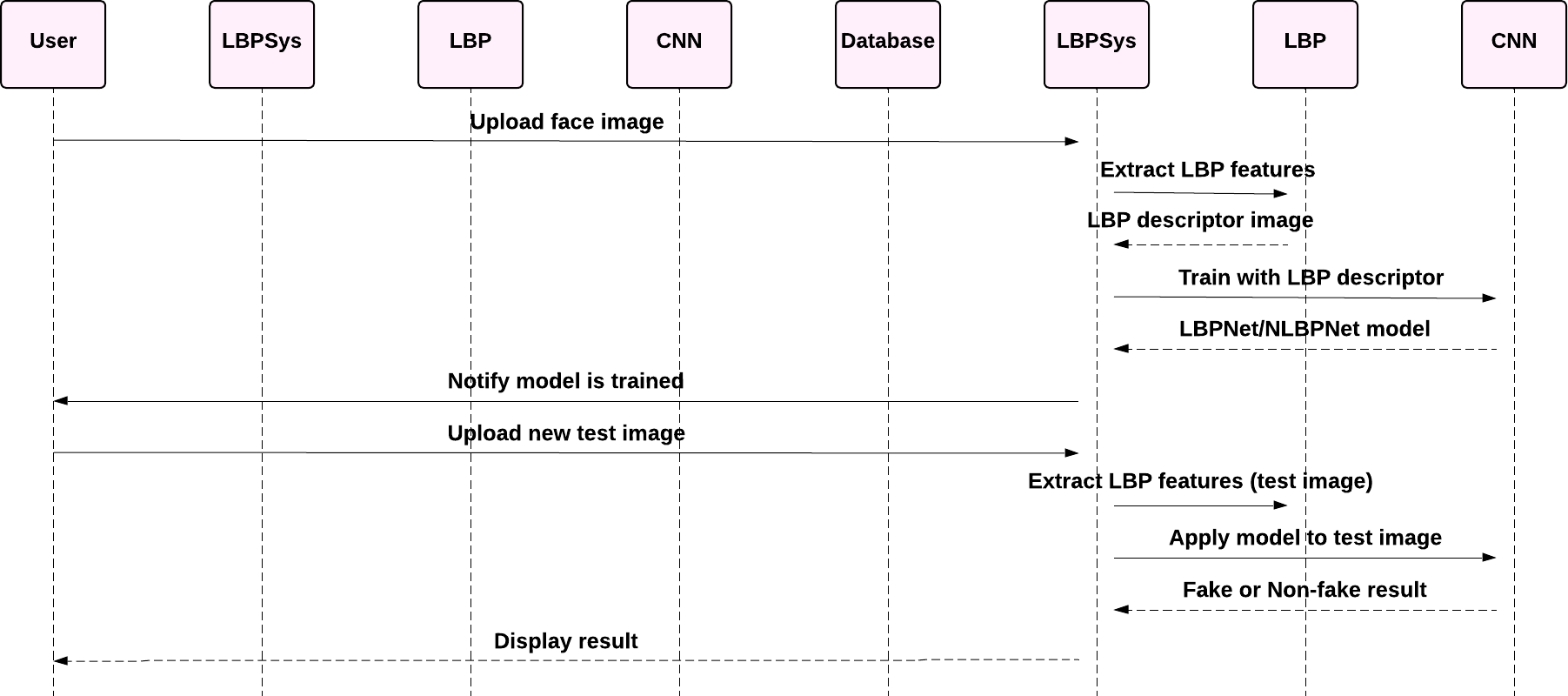
### Fig 4.3: Use Case Diagram

**Class Diagram:**



### Fig 4.4: Class Diagram

**Sequence Diagram:**



**Fig 4.5: Sequence Diagram**

# CHAPTER – 5

**SOFTWARE DEVELOPMENT LIFE CYCLE**

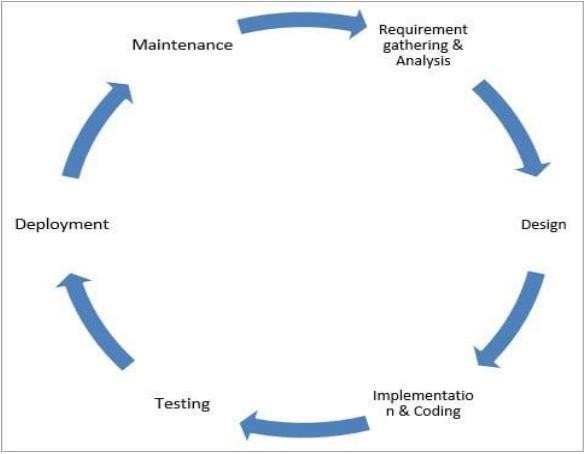
## 5.1 SDLC

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element in which life cycle model is followed; the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

The development team must determine a suitable life cycle model for a particular plan and then observe to it. Without using an exact life cycle model, the development of a software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team representatives about when and what to do. Otherwise, it would point to chaos and project failure. This problem can be defined by using an example. Suppose a software development issue is divided into various parts and the parts are assigned to the team members. From then on, suppose the team representative is allowed the freedom to develop the roles assigned to them in whatever way they like. One representative might start writing the code for his part, another might choose to prepare the test documents first, and some other engineer might begin with the design phase of the roles assigned to him. This would be one of the perfect methods for project failure.

A software life cycle model describes entry and exit criteria for each phase. A phase can begin only if its stage-entry criteria have been fulfilled. So, without a software life cycle model, the entry and exit criteria for a stage cannot be recognized. Without software life cycle models, it becomes tough for software project managers to monitor the progress of the project.



### Fig 5.1: Phases of SDLC Phase 1: Requirement collection and analysis

The requirement is the first stage in the SDLC process. It is conducted by the senior team

members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

### Phase 2: Defining Requirements

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted by the project stakeholders.

This is accomplished through the "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

### Phase 3: Designing the Software

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

### Phase 4: Developing the project

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

### Phase 5: Testing

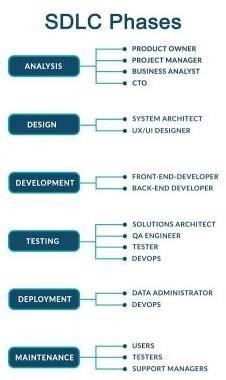
After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage. During this stage, unit testing, integration testing, system testing, acceptance testing are done.

### Phase 6: Deployment

Once the software is certified, and no bugs or errors are stated, then it is deployed. Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment. After the software is deployed, then its maintenance begins.

### Phase 7: Maintenance

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time. This procedure where the care is taken for the developed product is known as maintenance.



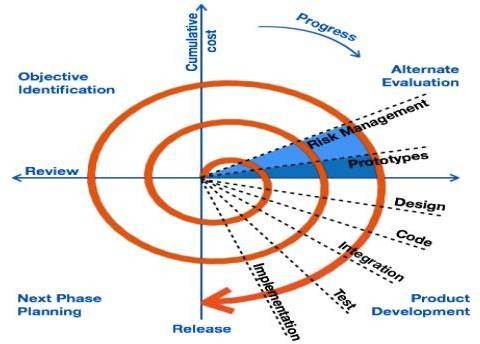
### Fig 5.2: Roles in SDLC

**Spiral Model Application**

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model −

* When there is a budget constraint and risk evaluation is important.
* For medium to high-risk projects.
* Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
* Customer is not sure of their requirements which is usually the case.
* Requirements are complex and need evaluation to get clarity.
* New product line which should be released in phases to get enough customer feedback. Significant changes are expected in the product during the development cycle.



**Fig 5.3: Spiral model of SDLC**

# CHAPTER 6 IMPLEMENTATION

## 6.1 SAMPLE CODE:

### Imageforgerydetection.py

#{'Fake': 0, 'Real': 1} from tkinter import \* import tkinter

from tkinter import filedialog import numpy as np

from tkinter.filedialog import askopenfilename import pandas as pd

from keras.optimizers import Adam

from keras.models import model\_from\_json from tkinter import simpledialog

from keras.models import Sequential from keras.layers import Convolution2D from keras.layers import MaxPooling2D from keras.layers import Flatten

from keras.layers import Dense,Activation,BatchNormalization import os

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator from tkinter import messagebox

import cv2

from imutils import paths import imutils

import cv2

import numpy as np main = tkinter.Tk()

main.title("Image Forgery Detection Using Deep Learning") #designing main screen main.geometry("600x500")

global filename global loaded\_model

def get\_pixel(img, center, x, y): new\_value = 0

try:

if img[x][y] >= center:

new\_value = 1 except:

pass

return new\_value

def lbp\_calculated\_pixel(img, x, y): center = img[x][y]

val\_ar = []

val\_ar.append(get\_pixel(img, center, x-1, y+1)) # top\_right val\_ar.append(get\_pixel(img, center, x, y+1)) # right val\_ar.append(get\_pixel(img, center, x+1, y+1)) # bottom\_right val\_ar.append(get\_pixel(img, center, x+1, y)) # bottom val\_ar.append(get\_pixel(img, center, x+1, y-1)) # bottom\_left val\_ar.append(get\_pixel(img, center, x, y-1)) # left val\_ar.append(get\_pixel(img, center, x-1, y-1)) # top\_left val\_ar.append(get\_pixel(img, center, x-1, y)) # top

power\_val = [1, 2, 4, 8, 16, 32, 64, 128]

val = 0

for i in range(len(val\_ar)):

val += val\_ar[i] \* power\_val[i] return val

def upload(): #function to upload tweeter profile global filename

filename = filedialog.askopenfilename(initialdir="testimages") messagebox.showinfo("File Information", "image file loaded")

def generateModel(): global loaded\_model

if os.path.exists('model.json'):

with open('model.json', "r") as json\_file:

loaded\_model\_json = json\_file.read()

loaded\_model = model\_from\_json(loaded\_model\_json)

loaded\_model.load\_weights("model\_weights.h5") loaded\_model.\_make\_predict\_function() print(loaded\_model.summary())

messagebox.showinfo("Model Generated", "CNN Model Generated on Train & Test Data. See black console for details")

else:

classifier = Sequential()

classifier.add(Convolution2D(32, (3, 3), border\_mode='valid', input\_shape=(48, 48, 1))) classifier.add(BatchNormalization())

classifier.add(Activation("relu")) classifier.add(Convolution2D(32, (3, 3), border\_mode='valid')) classifier.add(BatchNormalization()) classifier.add(Activation("relu"))

classifier.add(MaxPooling2D(pool\_size=(2, 2))) classifier.add(Flatten()) classifier.add(Dense(128)) classifier.add(BatchNormalization()) classifier.add(Activation("relu")) classifier.add(Dense(2)) classifier.add(BatchNormalization()) classifier.add(Activation("softmax"))

# model5 the model

classifier.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) files = []

filename = 'LBP/train/Forgery' label = []

for root, dirs, directory in os.walk(filename): for i in range(len(directory)):

files.append(filename+"/"+directory[i]); label.append([1,0])

filename = 'LBP/train/Real'

for root, dirs, directory in os.walk(filename): for i in range(len(directory)):

files.append(filename+"/"+directory[i]); label.append([0,1])

print(len(files))

X = np.ndarray(shape=(len(files), 48,48,1), dtype=np.float32) Y = np.ndarray(shape=(len(files),2),dtype=np.float32) print(X.shape)

print(Y.shape)

for i in range(len(files)):

img = cv2.imread(files[i])

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) img = np.resize(img, (48,48,1))

im2arr = np.array(img)

im2arr = im2arr.reshape(1,48,48,1) X[i] = im2arr

Y[i] = label[i]

print("shape == "+str(X.shape))

#X = X.reshape(X.shape[0],48, 48,3) classifier.fit(X, Y,epochs = 10) classifier.save\_weights('model\_weights.h5') model\_json = classifier.to\_json()

with open("model.json", "w") as json\_file: json\_file.write(model\_json)

print(X.class\_indices) print(classifier.summary)

messagebox.showinfo("Model Generated", "NLBPNet Model Generated on Train & Test Data. See black console for details")

def classify():

name = os.path.basename(filename) image\_file = filename;

img\_bgr = cv2.imread(image\_file) height, width, channel = img\_bgr.shape

img\_gray = cv2.cvtColor(img\_bgr, cv2.COLOR\_BGR2GRAY) img\_lbp = np.zeros((height, width,3), np.uint8)

for i in range(0, height):

for j in range(0, width):

img\_lbp[i, j] = lbp\_calculated\_pixel(img\_gray, i, j)

cv2.imwrite('testimages/lbp\_'+name, img\_lbp) img = cv2.imread('testimages/lbp\_'+name)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) img = np.resize(img, (48,48,1))

im2arr = np.array(img)

im2arr = im2arr.reshape(1,48,48,1) preds = loaded\_model.predict(im2arr)

print(str(preds)+" "+str(np.argmax(preds))) predict = np.argmax(preds)

msg = ""

if predict == 0:

msg = "Image Contains Forgery face Image" if predict == 1:

msg = "Image Contains Real face Image" imagedisplay = cv2.imread(filename)

orig = imagedisplay.copy()

output = imutils.resize(orig, width=400)

cv2.putText(output, msg, (10, 25), cv2.FONT\_HERSHEY\_SIMPLEX,0.7, (139, 0, 0), 2)

cv2.imshow("Predicted Image Result ", output) imagedisplay = cv2.imread('testimages/lbp\_'+name) orig = imagedisplay.copy()

output = imutils.resize(orig, width=400) os.remove('testimages/lbp\_'+name) cv2.imshow("LBP Image", output) cv2.waitKey(0)

def exit(): global main

main.destroy()

font = ('times', 16, 'bold')

title = Label(main, text='Image Forgery Detection Using Deep Learning', justify=LEFT) title.config(bg='lavender', fg='DarkOrchid1')

title.config(font=font) title.config(height=3, width=120) title.place(x=100,y=5)

title.pack()

font1 = ('times', 14, 'bold')

model = Button(main, text="Generate image Train & Test Model", command=generateModel) model.place(x=200,y=100)

model.config(font=font1)

uploadimage = Button(main, text="Upload Test Image", command=upload) uploadimage.place(x=200,y=150)

uploadimage.config(font=font1)

classifyimage = Button(main, text="Classify Picture In Image", command=classify) classifyimage.place(x=200,y=200)

classifyimage.config(font=font1)

exitapp = Button(main, text="Exit", command=exit) exitapp.place(x=200,y=250) exitapp.config(font=font1)

main.config(bg='light blue') main.mainloop()

# CHAPTER – 7 TESTING

## INTRODUCTION

Testing is a group of techniques to determine the correctness of the application under the predefined script but, testing cannot find all the defects of application. The main intent of testing is to detect failures of the application so that failures can be discovered and corrected. It does not demonstrate that a product functions properly under all conditions but only that it is not working in some specific conditions.

Testing includes an examination of code and also the execution of code in various environments, and conditions as well as all the examining aspects of the code. In the current scenario of software development, a testing team may be separate from the development team so that Information derived from testing can be used to correct the process of software development.

The success of software depends upon the acceptance of its targeted audience, easy graphical user interface, strong functionality load test, etc. For example, the audience of banking is totally different from the audience of a video game. Therefore, when an organization develops a software product, it can assess whether the software product will be beneficial to its purchasers and other audiences.

Types of manual testing:

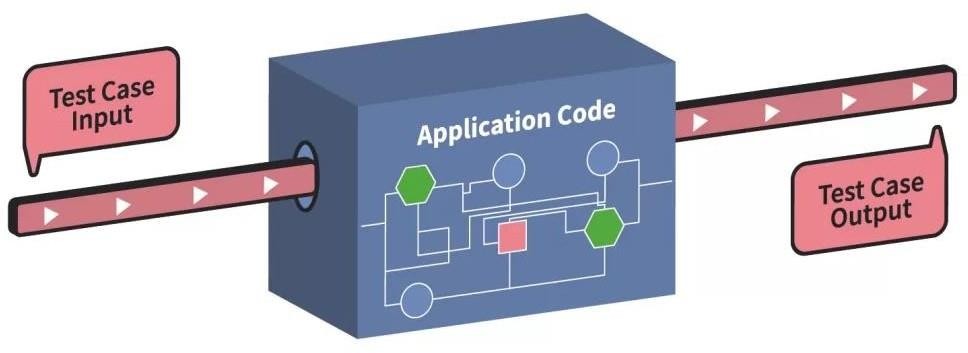
* White Box Testing
* Black Box Testing
* Gray Box Testing

### White Box Testing:

The box testing approach of software testing consists of black box testing and white box testing. White box testing which also known as glass box is testing, structural testing, clear box testing, open box testing, and transparent box testing. It tests the internal coding and infrastructure of a software focused on checking predefined inputs against expected and desired outputs. It is based on the inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthen the security of the software.

Generic steps of white box testing:

* Design all test scenarios, and test cases and prioritize them according to high-priority number.
* This step involves the study of code at runtime to examine the resource utilization, non- accessured areas of the code, time taken by various methods and operations and so on.
* In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, and interfaces can handle all types of data appropriately or not.
* This step focuses on testing control statements like loops and conditional statements to check the efficiency and accuracy of different data inputs.
* In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.



### Fig 7.1: White Box Testing

**Black box testing:**

Black box testing is a technique of software testing that examines the functionality of the software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that are stated by the customer.

In this method, the tester selects a function that gives input value to examine its functionality and checks whether the function is giving the expected output or not. If the function produces the correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Generic steps of black box testing:

* The black box test is based on the specification of requirements, so it is examined in the beginning.
* In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
* In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
* The fourth phase includes the execution of all test cases.
* In the fifth step, the tester compares the expected output against the actual output.
* In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.



### Fig 7.2: Black Box Testing

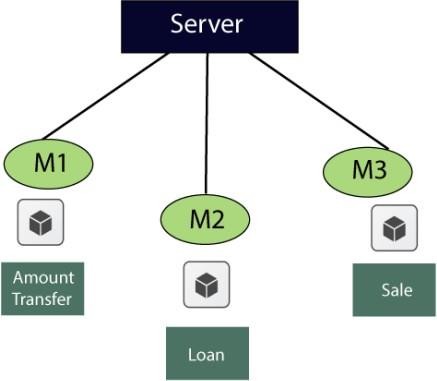
**Unit Testing:**

Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing. The aim behind unit testing is to validate unit components with their performance.

A unit is a single testable part of a software system and is tested during the development phase of the application software.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. The white box testing approach is used for unit testing and is usually done by the developers.

Whenever the application is ready and given to the Test engineer, he/she will start checking every component of the module or module of the application independently or one by one, and this process is known as Unit testing or components testing.



### Fig 7.3: Example of Unit Testing

**Integration Testing:**

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Unit testing uses modules for testing purposes, and these modules are combined and tested in integration testing. The Software is developed with several software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.



**Fig 7.4: Integration Testing**

## TEST CASES

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Expected Result** | **Obtained Result** |
| **TC\_001** | Input an authentic face image | Model predicts as authentic | Model predicts as authentic |
| **TC\_002** | Input a fake face image | Model predicts as fake | Model predicts as fake |
| **TC\_003** | Input a face image with Noise | Model performance degradation | Model performance degradation |
| **TC\_004** | Input a face image with compression artifacts | Model detects as fake | Model detects as fake |
| **TC\_005** | Input a face image with lighting variations | Model performance variation | Model performance variation |
| **TC\_006** | Input a face image with suitable manipulation | Model detects as fake | Model detects as fake |
| **TC\_007** | Input a face image with heavy manipulation | Model detects as fake | Model detects as fake |
| **TC\_008** | Input an authentic face image with watermark | Model detects as authentic | Model detects as authentic |
| **TC\_009** | Input a spliced face image | Model detects as fake | Model detects as fake |
| **TC\_010** | Input a face image with occlusions | Model performance degradation | Model performance degradation |

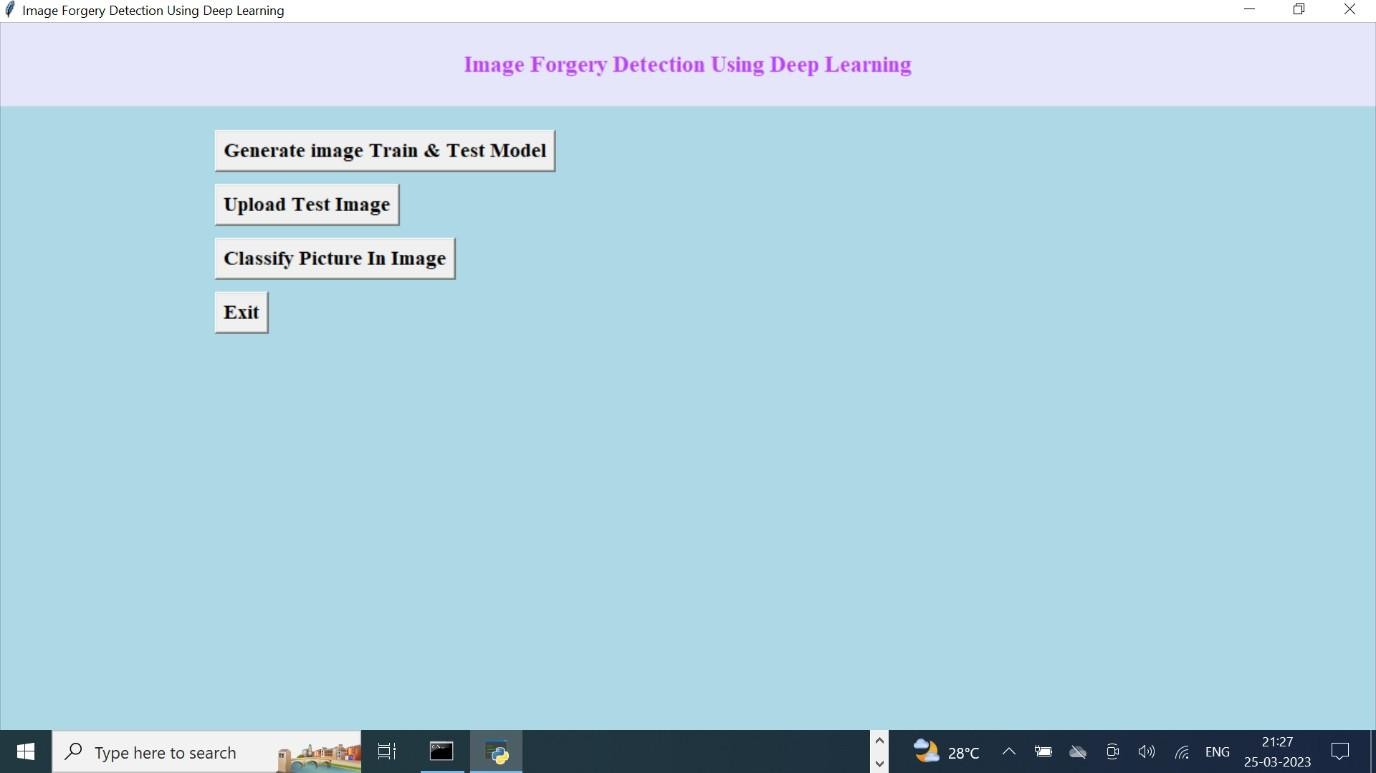
**Table 7.1: Test Cases**

# CHAPTER – 8

**INPUT & OUTPUT SCREENS**

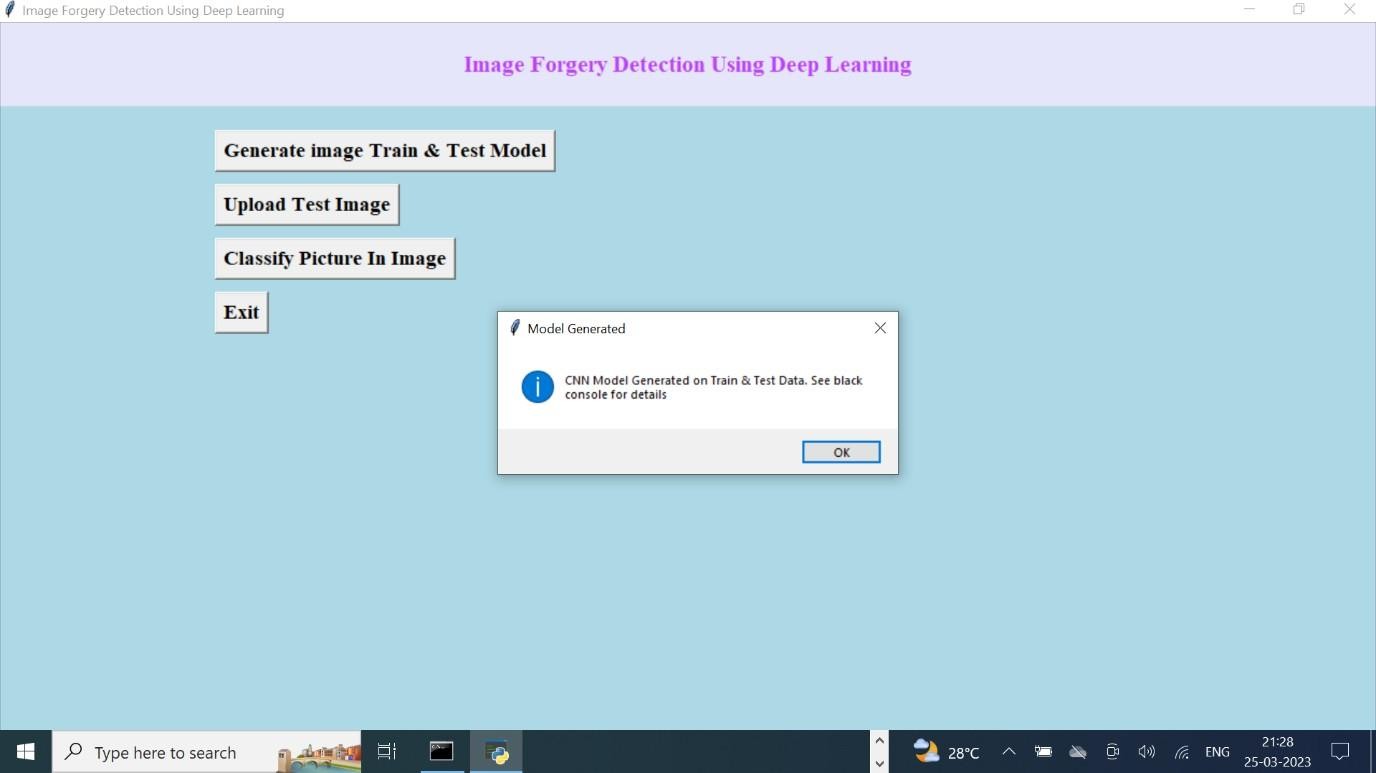
## INPUT SCREENS

To run this project double click on ‘run.bat’ file to get below screen.



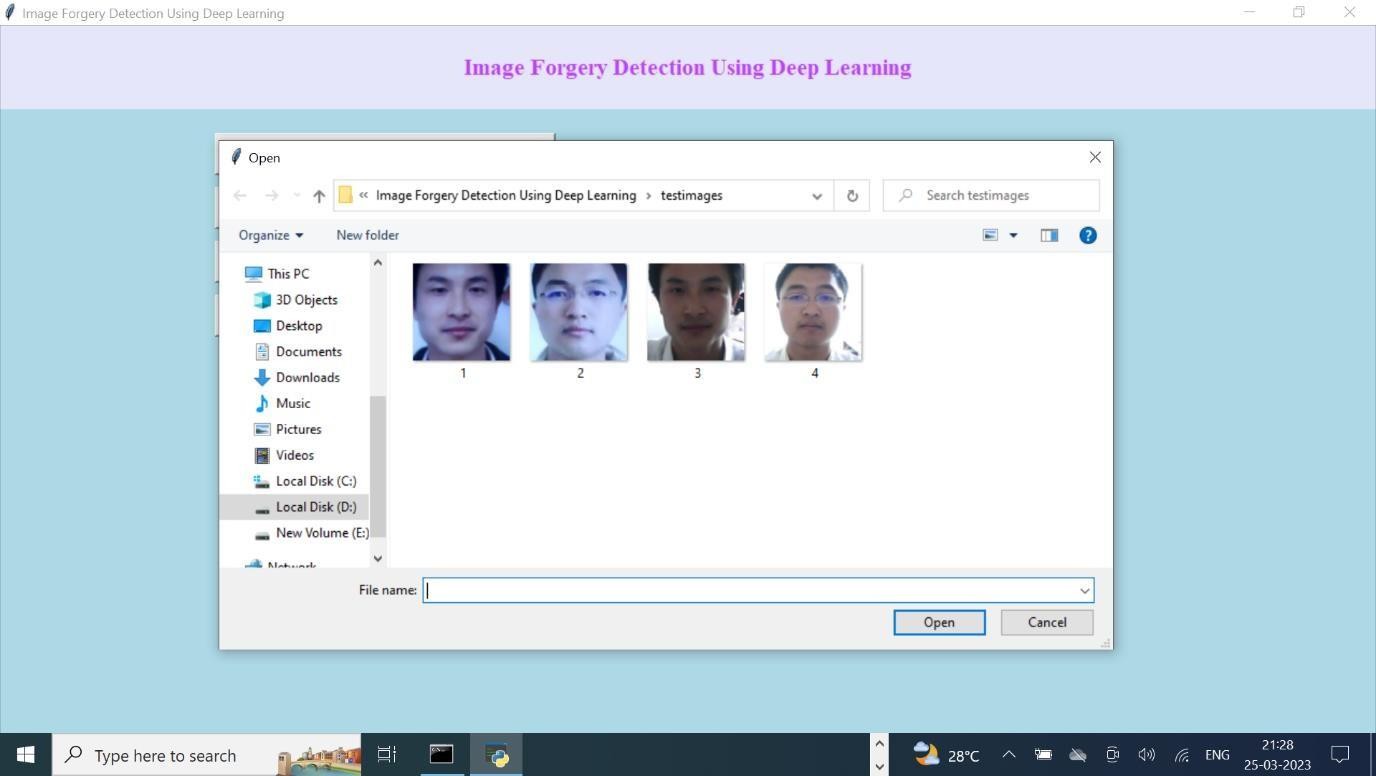
### Fig 8.1: Home Page of the application

In the above screen click on the ‘Generate Image Train & Test Model’ button to generate a CNN model using LBP images contains inside the LBP folder.



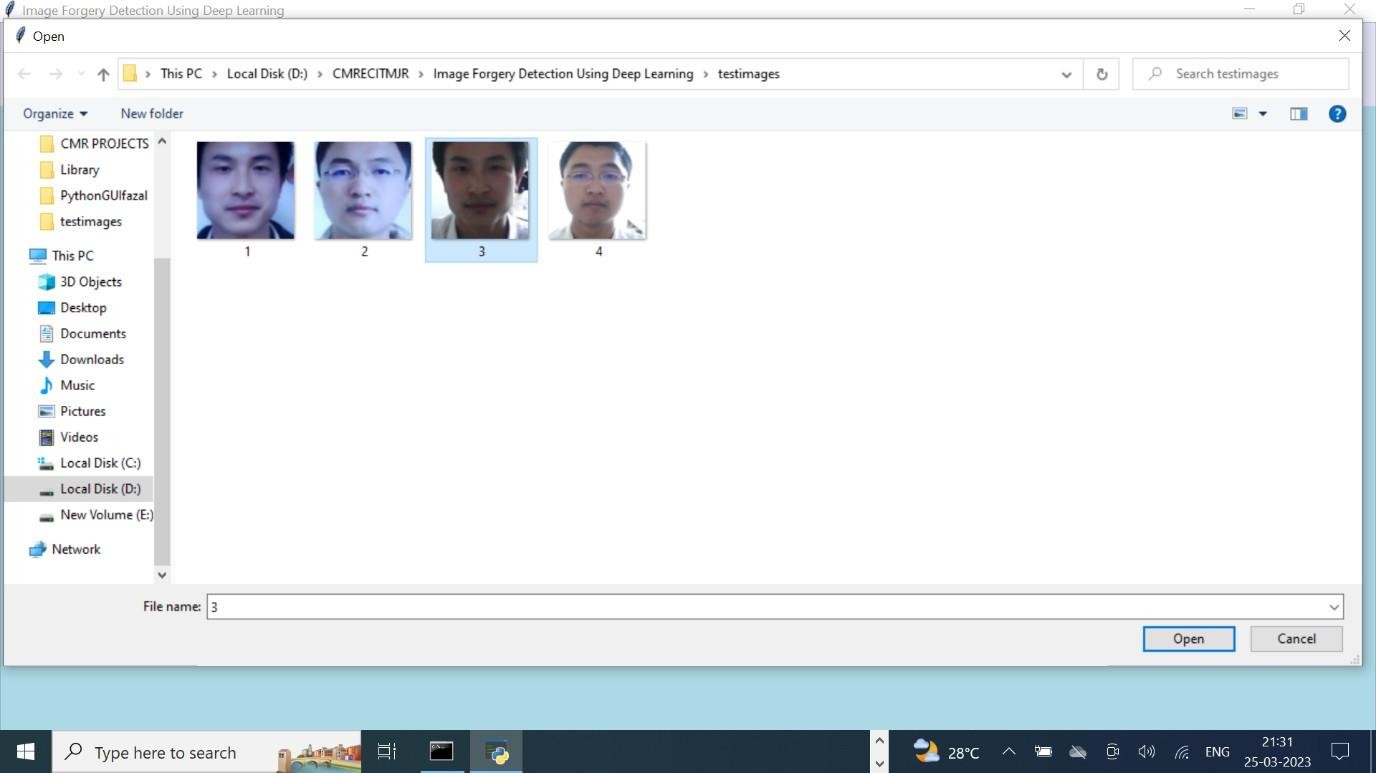
### Fig 8.2: Uploading Test Image

In the above screen, we can see the CNN LBPNET model generated. Now click on the ‘Upload Test Image’ button to upload a test image.



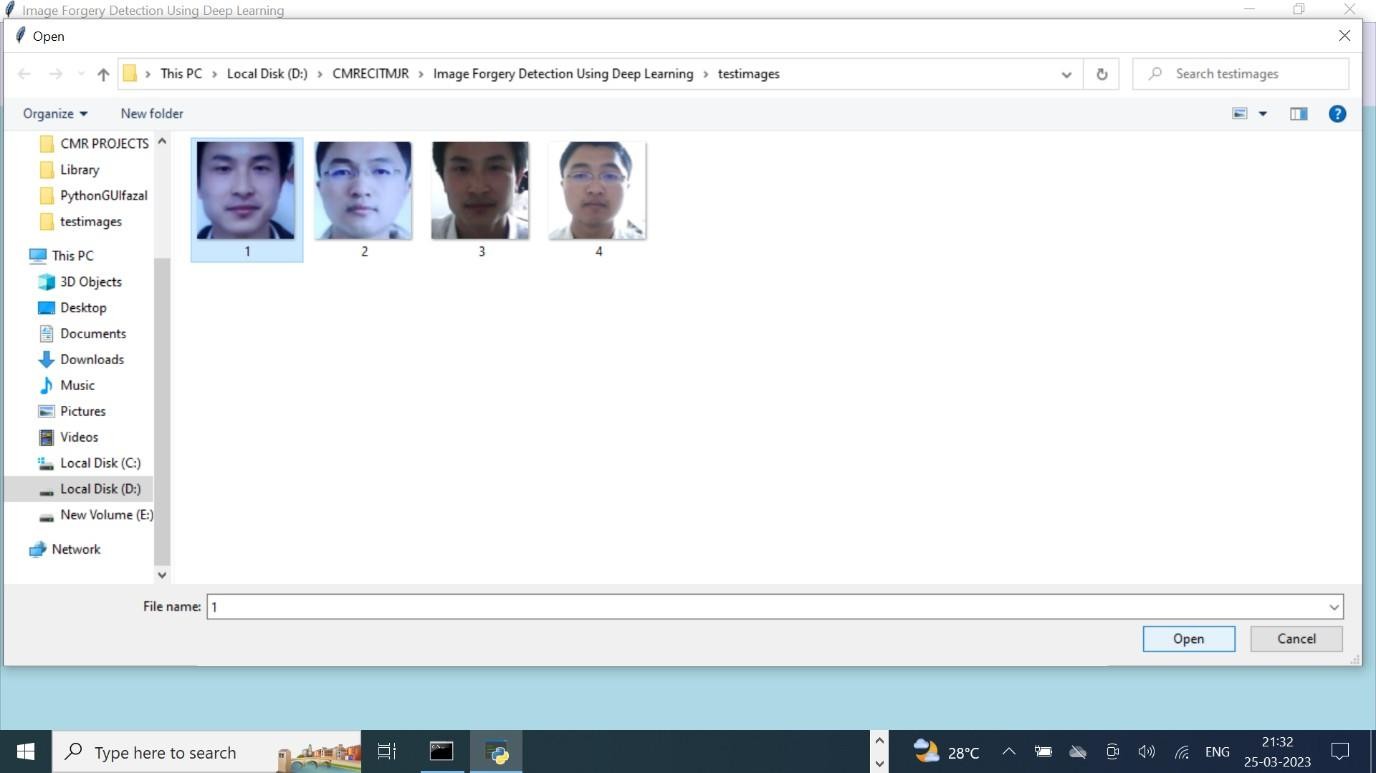
### Fig 8.3: Uploading Forgery Image

In the above screen, we can see two faces are there from the same person but in different appearances. For simplicity, we gave the image name as forgery and real to test whether the application can detect it or not. In the above screen, we uploaded a forgery image and then clicked on the ‘Classify Picture In Image’ button to get the below result.



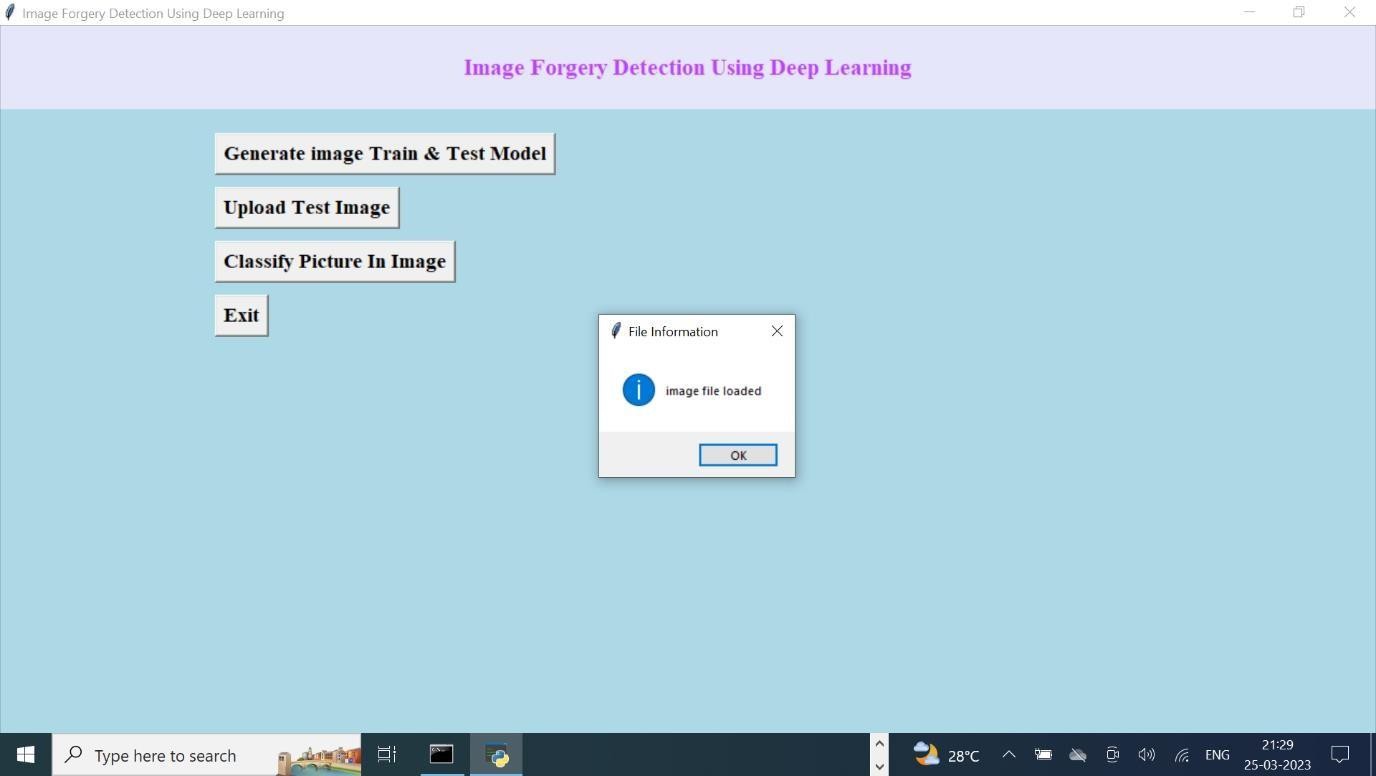
### Fig 8.4: Set of all the forged and original images

In the above screen, we can see all real faces will have normal light, and in forgery faces people will try some editing to avoid detection but this application will detect whether a face is real or forgery.



### Fig 8.5: Uploading Image 1

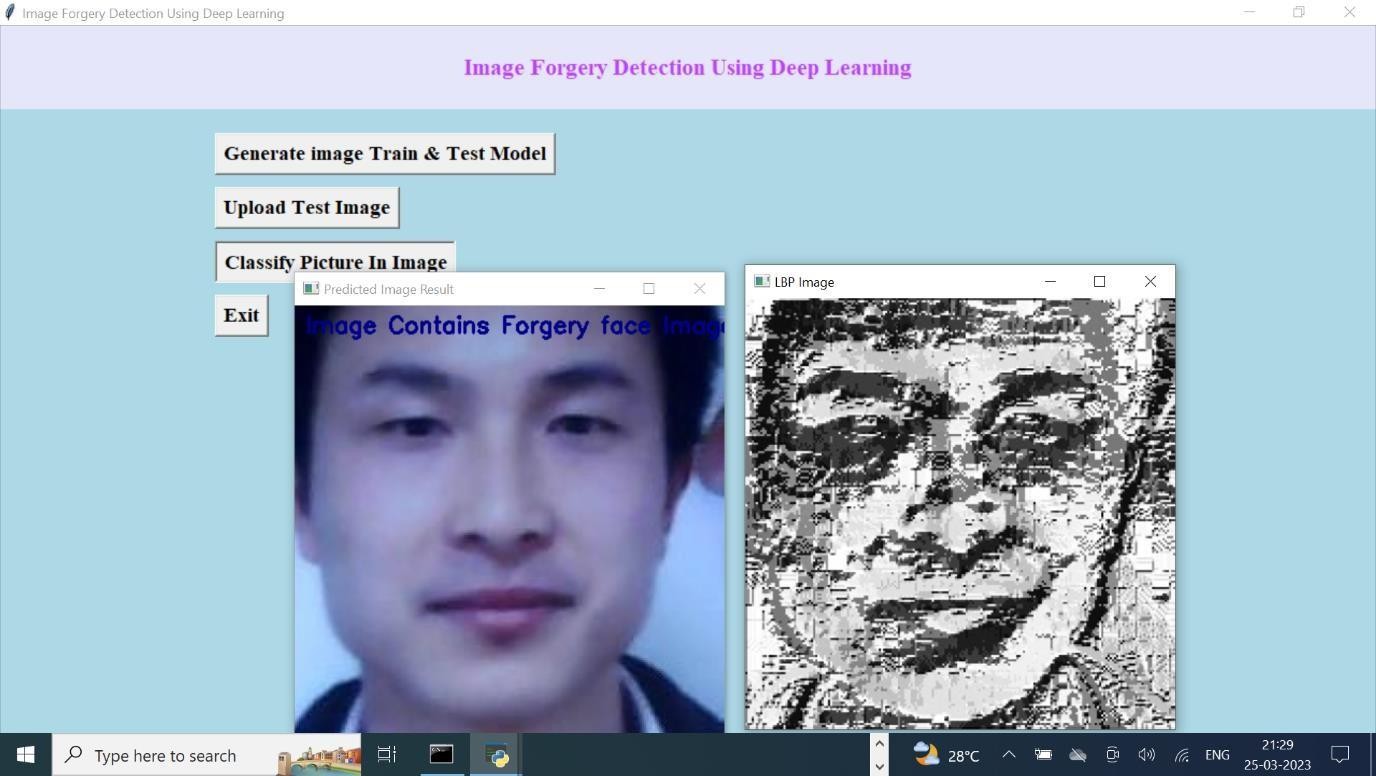
In the above screen, We uploaded 1.jpg, and after uploading click on the open button to get below screen.



### Fig 8.6: Click on Classify Button

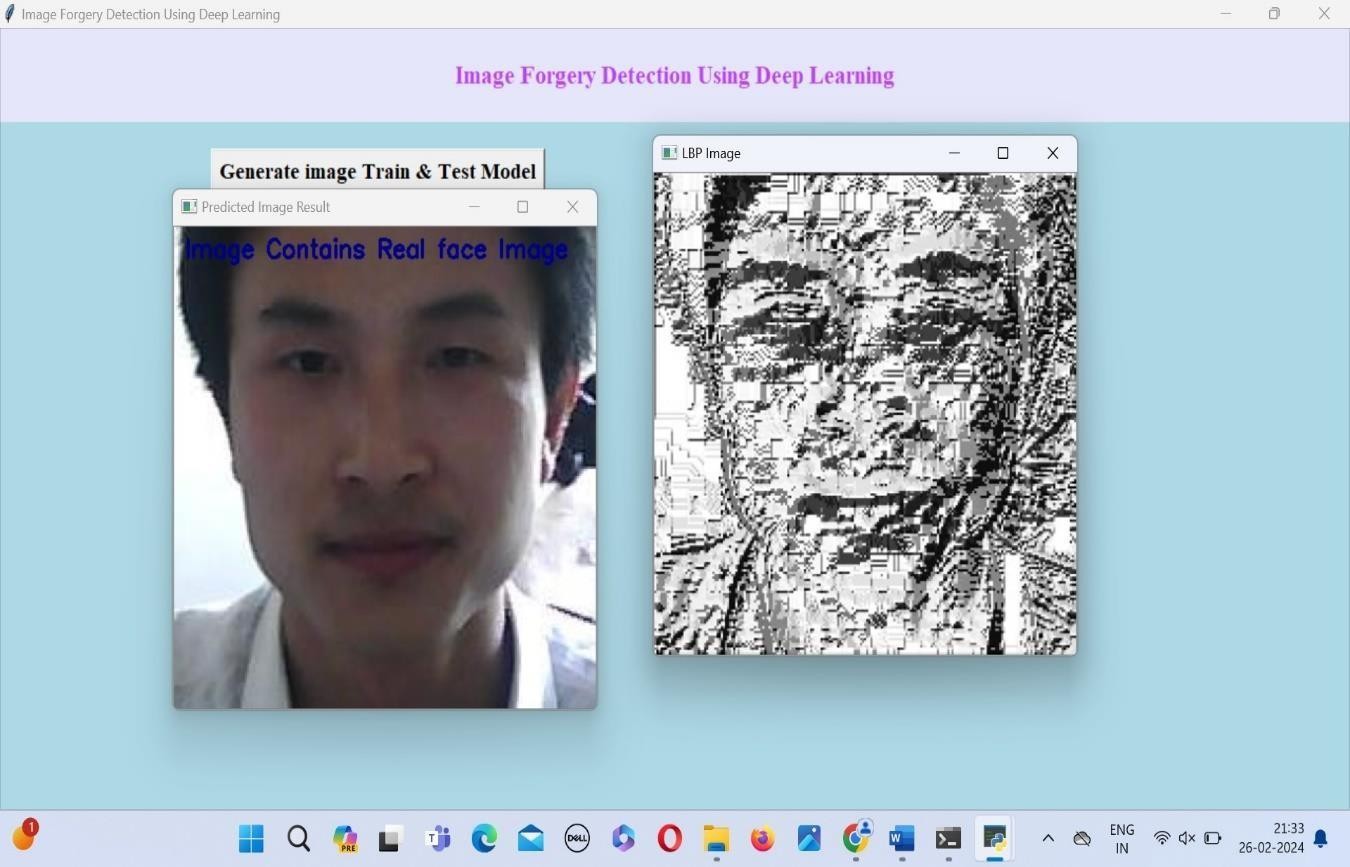
And now click on ‘Classify Picture in Image’ to get the below details.

## OUTPUT SCREENS



### Fig 8.7: Classified forgery image

In the above screen, we are getting results as the image contains a forgery face. Similarly, we can try other images also.



### Fig 8.8: Classified Original Image

In the above screen, we are getting the result as the image contains a real face. Similarly, we can try other images also.

# CHAPTER – 9 CONCLUSION AND FUTURE SCOPE

## CONCLUSION

In conclusion, the proposed forgery detection system, employing Local Binary Patterns (LBP) and Convolutional Neural Networks (CNNs) within the LBPNET architecture, offers a robust and effective solution for detecting fake face images. Unlike traditional methods that rely solely on pixel-level analysis or signature-based techniques, our approach leverages the rich texture information captured by LBP and the deep learning capabilities of CNNs to accurately identify subtle forgery patterns. By combining these techniques, our system achieves superior performance in distinguishing between authentic and manipulated face images, with high accuracy and reliability.

Moreover, the application of our forgery detection system extends beyond face images, offering versatility and adaptability to various forgery detection tasks in digital media. Its scalability, generalization, and ability to detect a wide range of forgery techniques make it a valuable tool for combating image tampering across different domains. Ultimately, our system enhances trust, credibility, and security in digital media and online platforms, providing a comprehensive solution to the growing challenges posed by image forgery in today's digital landscape.

## FUTURE SCOPE

### Semi-Supervised Learning:

Integrating semi-supervised learning techniques into the forgery detection system can leverage both labeled and unlabeled data to improve model performance. By utilizing the abundant unlabeled data available in real-world scenarios, semi-supervised approaches can enhance the model's ability to generalize and adapt to new forgery patterns.

### Adversarial Robustness:

Enhancing the robustness of the forgery detection system against adversarial attacks is crucial for real-world deployment. Future research can explore methods to defend against adversarial perturbations specifically tailored to fake face image detection, ensuring the model's reliability and effectiveness in adversarial environments.

### Multi-Modal Forgery Detection:

Extending the application to support multi-modal forgery detection, including audio and video modalities, can provide a more comprehensive solution for detecting multimedia forgeries. Integrating techniques such as audio analysis and video processing with the existing image forgery detection capabilities can enhance the system's versatility and applicability across diverse media types.

### Real-Time Detection:

Implementing real-time forgery detection capabilities can enable the system to analyze and detect fake face images in live streams or video feeds. This could find applications in surveillance, social media moderation, and online content verification, where timely detection of forgeries is critical for maintaining trust and security.

# CHAPTER – 10 REFERENCES

## 10.1 RESEARCH PAPERS

1. Sharma, S., & Yadav, S. (2019). Image forgery detection using error level analysis and deep learning, Research gate.
2. Rani, N., & Singh, P. (2022). A Comprehensive Survey on Image Forgery Detection Techniques. Advances in Intelligent Systems and Computing, 1435, 649-660.
3. Birajdar GK, Mankar VH (2013) Digital image forgery detection using passive techniques: a survey. Digit Investig 10(3):226–245. ( <https://doi.org/10.1016/j.diin.2013.04.007>)
4. Chen J, Liao X, Qin Z (2021) Identifying tampering operations in image operator chains based on decision fusion. Sig Process Image Commun 95:116287. <https://doi.org/10.1016/j.image.2021.116287>
5. Christlein V, Riess C, Angelopoulou E (2010) On rotation invariance in copy-move forgery detection. In: 2010 IEEE international workshop on information forensics and security, pp 1–6. <https://doi.org/10.1109/WIFS.2010.5711472>
6. Singh, G., & Chadha, A. (2020). Deep Learning-Based Image Forgery Detection: A Survey.

Journal of Imaging, 6(10), 103

1. Rahim, M. S. M., Wahab, A., & Idris, M. Y. I. (2017). Image Forgery Detection Using Convolutional Neural Network. 2017 International Conference on Engineering Technology and Technopreneurship (ICE2T).
2. Patel, K., & Patel, D. (2021). A Robust Image Forgery Detection Algorithm Using ELA and LBP. International Journal of Computer Applications in Technology, 66(3), 338-349.