**PICTURE DIFFERENCE RECOGNIZATION**

A Mini Project Report Submitted   
In partial fulfillment of the requirements for the award of the degree of

## Bachelor of Technology

## in

## Computer Science and Engineering

## by

## Kotha Monika 17N31A05B6

## Katkuri Sandeep Kumar 17N31A05A3

## Gogineni Venkata Ateet 17N31A0566

Under the esteemed guidance of

**Mrs. NIRMA DEVI**

**Associate Professor**



**Department of Computer Science and Engineering**

**Malla Reddy College of Engineering & Technology**

(Autonomous Institution- UGC, Govt. of India)

(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA &NAAC with ‘A’ Grade)

Maisammaguda, Kompally, Dhulapally, Secunderabad – 500100  
website: www.mrcet.ac.in   
2020-2021



**Malla Reddy College of Engineering & Technology**

(Autonomous Institution- UGC, Govt. of India)

(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA &NAAC with ‘A’ Grade)

Maisammaguda, Kompally, Dhulapally, Secunderabad – 500100  
website: www.mrcet.ac.in

**CERTIFICATE**

This is the bonafide record of the project entitled “Picture Difference Recognition”, submitted by Kotha Monika (17N31A05B6), Katkuri Sandeep Kumar (17N31A05A3) and Gogineni Venkata Ateet sandeep (17N31A0566) of B.Tech in the partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science and Engineering, Department of CSE during the year 2020-2021. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

**Internal Guide Project Co-ordinator**

**Mrs. Nirma Devi Dr. S. Shanthi**

**Assistant Professor Professor**

**Head of the Department EXTERNAL EXAMINER**

**Dr. D. Sujatha**

**Professor**

**DECLARATION**

## We hereby declare that the project titled “Image Difference Recognition” submitted to Malla Reddy College of Engineering and Technology (UGC Autonomous), affiliated to Jawaharlal Nehru Technological University Hyderabad (JNTUH) for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a result of original research carried-out in this thesis. It is further declared that the project report or any part thereof has not been previously submitted to any University or Institute for the award of degree or diploma.

## Kotha Monika 17N31A05B6

## Katkuri Sandeep Kumar 17N31A05A3

## Gogineni Venkata Ateet 17N31A0566

# ACKNOWLEDGEMENT

We feel ourselves honored and privileged to place our warm salutation to our college Malla Reddy College of Engineering and Technology (UGC-Autonomous) and our Principal **Dr. VSK Reddy** who gave us the opportunity to have experience in engineering and profound technical knowledge.

We express our heartiest thanks to our Head of the Department **Dr. D. Sujatha** for encouraging us in every aspect of our project and helping us realize our full potential.

We would like to thank our internal guide **Mrs. Nirma Devi**, Assistant Professor and Project Coordinator **Dr.S.Shanthi**, for their regular guidance and constant encouragement. We are extremely grateful to her valuable suggestions and unflinching co-operation throughout project work.

We would like to thank our Class Incharge Dr.M.Jayapal who in spite of being busy with his duties took time to guide and keep us on the correct path.

We would also like to thank all the supporting staff of the Department of CSE and all other departments who have been helpful directly or indirectly in making our project a success.

We are extremely grateful to our parents for their blessings and prayers for the completion of our project that gave us strength to do our project.

With regards and gratitude,

## Kotha Monika - 17N31A05B6

## Katkuri Sandeep Kumar - 17N31A05A3

**Gogineni Venkata Ateet - 17N31A0566**

**ABSTRACT**

**Picture differencing** is a image processing technique used to determine changes between images. The difference between two images is calculated by finding the difference between each pixel in each image, and generating an image based on the result. For this technique to work, the two images must first be aligned so that corresponding points coincide, and the values must be made compatible, either by careful calibration, or by post-processing.

**Software Requirements are:**

* Python
* Microsoft visual studio code / PyCharm
* Operating System can be Windows XP and above.

**Hardware Requirements are:**

* Ram: 1GB Ram and above
* Hard Disk: 50GB and above
* Processor: Dual core and above
* OpenCV will be downloaded and used

To run this Application, we need Win or Linux OS PCs

|  |  |
| --- | --- |
|  | **CONTENTS PAGE NO**  **1. INTRODUCTION TO PROJECT**  1.1 PURPOSE OF THE PROJECT 01  1.2 SCOPE OF PROJECT 01  1.3 FEATURES OF THE PROJECT 01 |
|  | **2. SYSTEM ANALYSIS AND DESCRIPTION**  2.1 EXISTING SYSTEM 09  2.2 PROPOSED SYSTEM 09  2.3 FEASIBILITY STUDY 09  2.4 SDLC 10 - 13 |
|  | **3. SYSTEM DESIGN**  3.1 SOFTWARE DESIGN 41  3.2 INPUT/ OUTPUT DESIGN 15  3.3 SOFTWARE REQUIREMENT SPECIFICATION 15  3.4 DOCUMENT CONVENTIONS 17  3.5 INTENDED AUDIENCE AND READING CONVEN TIONS 17  3.6 UML CONCEPTS 17 - 27  **4**. **IMPLEMENTATION**  4.1 PYTHON 28  4.2 THE HISTORY OF PYTHON 29  4.3 CODING 30  4.4 IMPLEMENTATION 31  4.5 SOFTWARE TOOL 31 - 32 |
|  | **5. TESTING**  5.1 SOFTWARE TESTING 33  5.2 TEST CASE DESIGN 33  5.3 TEST CASES 34  **6. SCREEN SHOTS** 35 - 39  **7. CONCLUSION** 40  **8. BIBILOGRAPHY** 40 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE.NO** | **NAME** | **PAGE.NO** |
| FIG 2.4 | SDLC | 13 |
| FIG 3.1.1 | SYSTEM ARCHITECTURE DESIGN | 14 |
| FIG 3.6.1 | 1. THINGS 2. RELATIONSHIPS 3. SEQUENCE DIAGRAMS | 18  20  21 |
| FIG 3.6.2.1 | CLASS DIAGRAM | 24 |
| FIG 3.6.2.2 | USECASE DIAGRAM | 25 |
| FIG 3.6.2.3 | ACTIVITY DIAGRAM | 26 |
| FIG 3.6.2.4 | STATECHART DIAGRAM | 27 |

­­­­

**Chapter 1: INTRODUCTION**

**1.1 PURPOSE OF THE PROJECT**

The purpose of the project is to differentiate the pictures. In some situations, it is difficult to find difference of pictures when there are minor changes. So, this program will help us to find the difference in the pictures by showing them in a rectangular box. It takes less time complexity and we can spot each and every difference between the pictures.

**1.2 SCOPE OF THE PROJECT**

The scope of the project is to differentiate what is original or what is duplicate by comparison. The thing which cannot done by human eye, will be done by this program. By this we can spot easily what difference between the pictures.

**1.3 FEATURES OF THE PROJECT**

1: It differentiates the two pictures even it is a slight difference.

2: It decreases the time complexity of users to find difference.

3: Results shown in rectangular box and thresholds

**Chapter 2: System Analysis and Description**

**2.1 EXISTING SYSTEM**

Here we've got applied the SSIM which is method for predicting cinematic pictures, digital images and videos etc.

By this we are able to cluster the photographs therefore we are able to find the difference by parting the images.

**2.2 PROPOSED SYSTEM**

Using this method, we were able to easily determine if two images were identical or had differences due to slight image manipulations, compression artifacts, or purposeful tampering.

Developing a phishing detection system is obviously much more complicated than simple image differences, but we can still apply these techniques to determine if a given image has been manipulated.

**2.2.1 Advantages**

1. it differentiates the two pictures even it is a slight difference.

2. it decreases the time complexity of users to find difference.

3. results shown in rectangular box and thresholds

2.3 FEASIBILITY STUDY**:**

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

Three key considerations involved in the feasibility analysis are

• Economic Feasibility

• Technical Feasibility

• Social Feasibility

## 2.4.1 Technical Feasibility:

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

**2.4.2** Social Feasibility:

The aspect of study is to check the level of acceptance of the system by the employee. This includes the process of responding the employee to his request efficiently. The employee must not feel hesitated, instead must accept it as a necessity.

## 2.4.3 Economic Feasibility:

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the organization can pour into the research and development of the system is limited. The expenditures must be justified.

**2.4 SDLC (SOFTWARE DEVELOPMENT LIFE CYCLE)**

**2.5.1 SDLC Methodologies:**

This document plays a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

**2.5.2 SPIRAL MODEL:**

It was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development. The spiral model is similar to the [incremental model](http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/), with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spirals, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

* The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:

1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
2. Defining the requirements of the second prototype.
3. Planning a designing the second prototype.
4. Constructing and testing the second prototype.

* At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.
* The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype.
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

**The following diagram shows how a spiral model acts like:**



**Fig.2.5 Spiral Model**

* **Planning Phase:**Requirements are gathered during the planning phase. Requirements like ‘BRS’ that is ‘Business Requirement Specifications’ and ‘SRS’ that is ‘System Requirement specifications’.
* **Risk Analysis:** In the**risk analysis phase**, a [process](http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/) is undertaken to identify risk and alternate solutions.  A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.
* **Engineering Phase:** In this phase software is **developed**, along with [testing](http://istqbexamcertification.com/what-is-a-software-testing/) at the end of the phase. Hence in this phase the development and testing is done.
* **Evaluation phase:**This phase allows the customer to evaluate the output of the project to [date](http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/) before the project continues to the next spiral

**Chapter 3: SYSTEM DESIGN**

System design is transition from a user-oriented document to programmers or data base personnel. The design is a solution, how to approach to the creation of a new system. This is composed of several steps. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Designing goes through logical and physical stages of development, logical design reviews the present physical system, prepare input and output specification, details of implementation plan and prepare a logical design walkthrough.

**3.1 SOFTWARE DESIGN**

In designing the software following principles are followed:

* **Modularity and partitioning**: software is designed such that, each system should

consist of hierarchy of modules and serve to partition into separate function.

* **Coupling:** modules should have little dependence on other modules of a system.
* **Cohesion:** modules should carry out in a single processing function.
* **Shared use:** avoid duplication by allowing a single module be called by other that need the function it provides

**3.1.1 System Architecture Design**

Covert into gray scale

Image comparison

ROI

Compute SSIM

Input image

Nndn

Fig.3.1.1 System Architecture Design

**3.2.1 Input Design:**

Considering the requirements, procedures to collect the necessary

input data in most efficiently designed. The input design has been done keeping in view that, the interaction of the user with the system being the most effective and simplified way.

Also, the measures are taken for the following

* + - Eliminating extra steps
    - Keeping the process simple

**3.2.2 Output Design:**

The output contains the actual image differences between two input images that we wish to visualize.

The rectangles are drawn around the different regions of each image. We show the comparisons images with boxes around differences, the difference image and the threshold image.

**3.3 SOFTWARE REQUIREMENT SPECIFICATION**

**3.3.1What is SRS?**

A **software requirements specification** (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system. The software requirement specification document consistent of all necessary requirements required for project development. To develop the software system, we should have clear understanding of Software system. To achieve this, we need to continuous communication with customers to gather all requirements.

1. **Correctness of SRS should be checked.**Since the whole testing phase is dependent on SRS, it is very important to check its correctness. There are some standards with which we can compare and verify.
2. **Ambiguity should be avoided.**Sometimes in SRS, some words have more than one meaning and this might confuse testers making it difficult to get the exact reference. It is advisable to check for such ambiguous words and make the meaning clear for better understanding.
3. **Requirements should be complete.**When tester writes test cases, what exactly is required from the application, is the first thing which needs to be clear. For e.g. if application needs to send the specific data of some specific size then it should be clearly mentioned in SRS that how much data and what is the size limit to send.
4. **Consistent requirements:** The SRS should be consistent within itself and consistent to its reference documents. If you call an input “Start and Stop” in one place, don’t call it “Start/Stop” in another. This sets the standard and should be followed throughout the testing phase.
5. **Verification of expected result:**SRS should not have statements like “Work as expected”, it should be clearly stated that what is expected since different testers would have different thinking aspects and may draw different results from this statement.
6. **Testing environment:** some applications need specific conditions to test and also a particular environment for accurate result. SRS should have clear documentation on what type of environment is needed to set up.
7. **Pre-conditions defined clearly:**one of the most important part of test cases is pre-conditions. If they are not met properly then actual result will always be different expected result. Verify that in SRS, all the pre-conditions are mentioned clearly.
8. **Requirements ID:** these are the base of test case template. Based on requirement Ids, test case ids are written. Also, requirements ids make it easy to categorize modules so just by looking at them, tester will know which module to refer. SRS must have them such as id defines a particular module.
9. **Security and Performance criteria:**security is priority when a software is tested especially when it is built in such a way that it contains some crucial information when leaked can cause harm to business. Tester should check that all the security related requirements are properly defined and are clear to him. Also, when we talk about performance of a software, it plays a very important role in business so all the requirements related to performance must be clear to the tester and he must also know when and how much stress or load testing should be done to test the performance.
10. **Assumption should be avoided:**sometimes when requirement is not cleared to tester, he tends to make some assumptions related to it, which is not a right way to do testing as assumptions could go wrong and hence, test results may vary. It is better to avoid assumptions and ask clients bout all the “missing requirements” to have a better understanding of expected results.
11. **Deletion of irrelevant requirements:**there are more than one team who work on SRS so it might be possible that some irrelevant requirements are included in SRS. Based on the understanding of the software, tester can find out which are these requirements and remove them to avoid confusions and reduce work load.
12. **Freeze requirements:** when an ambiguous or incomplete requirement is sent to client to analyse and tester gets a reply, that requirement result will be updated in the next SRS version and client will freeze that requirement. Freezing here means that result will not change again until and unless some major addition or modification is introduced in the software.

The SRS phase consists of two basic activities:

**3.3.1.1 Problem/Requirement Analysis:**

The process is order and more nebulous of the two, deals with understand the problem, the goal and constraints.

**3.3.1.2 Requirement Specification:**

Here, the focus is on specifying what has been found giving analysis such as representation, specification languages and tools, and checking the specifications are addressed during this activity.

The Requirement phase terminates with the production of the validate SRS document. Producing the SRS document is the basic goal of this phase.

**3.4 DOCUMENT CONVENTIONS**

We have used Times New Roman (text size 12). Bold Font is used for Main Headings (text size of 16). Normal font is used for sub headings (text size of 14).

**Font:** Times New Roman

**Main Heading:** Bold Font

## 3.5 INTENDED AUDIENCE AND READING SUGGESTIONS

This document is for better understanding for Picture difference recognition. Mainly intended for Head of the Dept., Internal guide, External guide, Staff members, Users and colleagues. This detail given below guides every normal user to how to go through this document for better understanding. The sequence to follow for better understanding is here Purpose, Scope, Features, Operating requirements, Modules present in the project, Advantages, References etc.

**3.5.1 Role of SRS:**

The purpose of the Software Requirement Specification is to reduce the communication gap between the clients and the developers. Software Requirement Specification is the medium though which the client and user needs are accurately specified. It forms the basis of software development. A good SRS should satisfy all the parties involved in the system.

## 3.5.2 Scope:

The picture difference recognition is for finding the difference between the two images. This helps in computing

3.6 UML CONCEPTS:

The Unified Modeling Language (UML) is a standard language for writing software blue prints. The UML is a language for

* Visualizing
* Specifying
* Constructing
* Documenting the artifacts of a software intensive system.

The UML is a language which provides vocabulary and the rules for combining words in that vocabulary for the purpose of communication. A modeling language is a language whose vocabulary and the rules focus on the conceptual and physical representation of a system. Modeling yields an understanding of a system.

3.6.1 Building Blocks of the UML:

The vocabulary of the UML encompasses three kinds of building blocks:

* Things
* Relationships
* Diagrams

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

3.6.1.1 Things in the UML:

There are four kinds of things in the UML:

* Structural things
* Behavioral things
* Grouping things
* Annotational things
* **Structural things** are the nouns of UML models. The structural things used in the project design are first, a **class** is a description of a set of objects that share the same attributes, operations, relationships and semantics.

|  |
| --- |
| Window |
| Origin  Size |
| open()  close()  move()  display() |

##### 

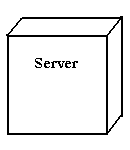
##### Fig: Classes

Second, a **use case** is a description of set of sequence of actions that a system performs that yields an observable result of value to particular actor.



**Fig: Use Cases**

Third, a node is a physical element that exists at runtime and represents a computational resource, generally having at least some memory and often processing capability.



**Fig: Nodes**

* **Behavioral things** are the dynamic parts of UML models. The behavioral thing used is:

**Interaction:**

An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. An interaction involves a number of other elements, including messages, action sequences (the behavior invoked by a message, and links (the connection between objects).



**Fig: Messages**

**3.6.1.2 Relationships in the UML:**

There are four kinds of relationships in the UML:

* Dependency
* Association
* Generalization
* Realization

A **dependency** is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing (the dependent thing).

**Fig: Dependencies**

An **association** is a structural relationship that describes a set links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts.



**Fig: Association**

A **generalization** is a specialization/ generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent).



**Fig: Generalization**

A **realization** is a semantic relationship between classifiers, where in one classifier specifies a contract that another classifier guarantees to carry out.



**Fig: Realization**

**3.6.1.3 Sequence Diagrams:**

UML sequence diagrams are used to represent the flow of messages, events and actions between the objects or components of a system. Time is represented in the vertical direction showing the sequence of interactions of the header elements, which are displayed horizontally at the top of the diagram.

Sequence Diagrams are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario. UML sequence diagrams are useful design tools because they provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications.

* **Actor**

Represents an external person or entity that interacts with the system

**Sequence diagram actor element**

* **Object**

Represents an object in the system or one of its components

**Sequence diagram object element**

* **Unit**

Represents a subsystem, component, unit, or other logical entity in the system (may or may not be implemented by objects)

**Sequence diagram unit element**

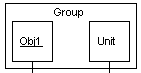
* **Separator**

Represents an interface or boundary between subsystems, components or units (e.g., air interface, Internet, network)

**Sequence diagram separator element**

* **Group**

Groups related header elements into subsystems or components

****

## 3.6.1.4Sequence Diagram Body Elements

* **Action**

Represents an action taken by an actor, object or unit

**Sequence diagram action element**

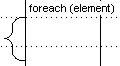
* **Asynchronous Message**

An asynchronous message between header elements

**Sequence diagram asynchronous message element**

* **Block**

A block representing a loop or conditional for a particular header element

****

* **Call Message**

A call (procedure) message between header elements

**Sequence diagram call message element**

* **Create Message**

A "create" message that creates a header element (represented by lifeline going from dashed to solid pattern)

**Sequence diagram create message element**

* **Diagram Link**

Represents a portion of a diagram being treated as a functional block. Similar to a procedure or function call that abstracts functionality or details not shown at this level. Can optionally be linked to another diagram for elaboration.

**Sequence diagram diagram link element**

Else Block Represents an "else" block portion of a diagram block

**Sequence diagram else block element**

* **Message**

A simple message between header elements

**Sequence diagram message element**

* **Return Message**

A return message between header element.

**UML DIAGRAMS**

**Class:**

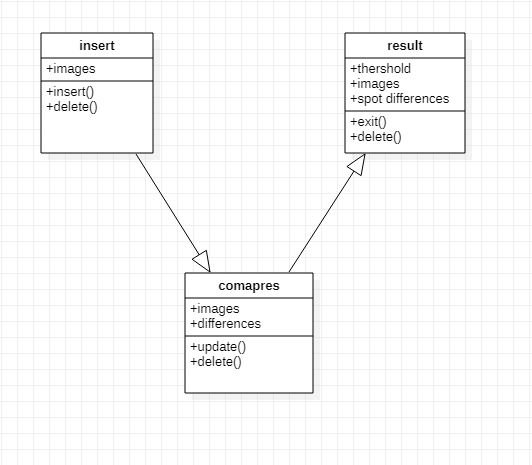


Fig.3.6.2.1 Class Diagram

**Class** is a description of a set of objects that share the same attributes, operations, relationships and semantics. For an instance, Menu is a class. It has attributes like time schedule, view time, exit and operations like edit, update and cancel.

**Use case:**

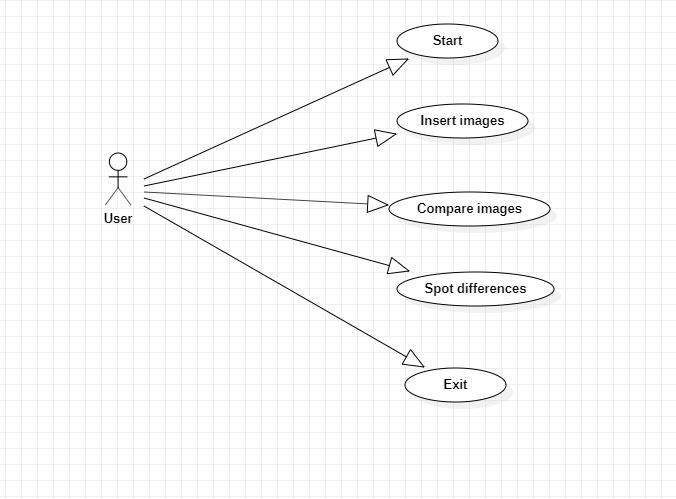


Fig.3.6.2.2 Use Case Diagram

**Use Case** is a description of set of sequence of actions that a system performs that yields an observable result of value to particular actor. Here, the sequence of actions is registration, login, creating time schedule, view time schedule and exit.

**Activity:**

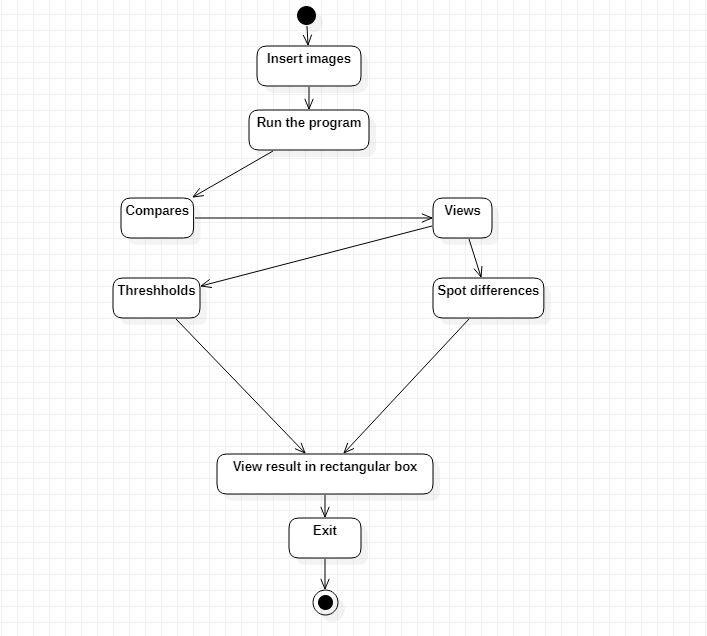


Fig.3.6.2.3 Activity Diagram

**Activity Diagram** shows the sequence of activities that takes place in the application. It starts with the registration of the user, followed by login. It follows the forking of time schedule into create, view, which are further forked into starting time, ending time, edit, update. These are joined into save details activity. It is, then followed by exit activity.

**State chart:**

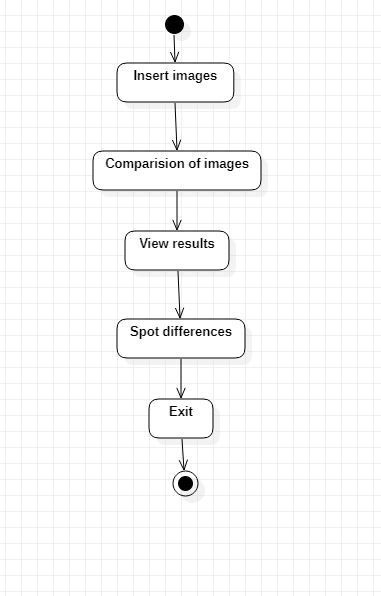


Fig.3.6.2.4 State Chart Diagram

**State Chart Diagram** has a sequence of steps as registration, login, create timings, view timings and exit.

**Chapter4 : IMPLEMENTATION**

**4.1 PYTHON**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

# 4.2 THE HISTORY OF PYTHON

Python was conceived in the late 1980s[35] by Guido van Rossum at Centrum Wickenden & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's Benevolent Dictator for Life, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. He now shares his leadership as a member of a five-person steering council. In January 2019, active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw, Carol Willing and Van Rossum to a five-member "Steering Council" to lead the project.

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6. and 2.7. version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system with reference counting.

Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

The Python 2 language was officially discontinued in 2020 (first planned for 2015), and "Python 2.7.18 is the last Python 2.7 release and therefore the last Python 2 release." No more security patches or other improvements will be released for it. With Python 2's end-of-life, only Python 3.5. and later are supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains Python, a free and open-source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python development.

# 4.3 CODING:

# USAGE

# python image\_diff.py --first images/original\_01.png --second images/modified\_01.png

# import the necessary packages

from skimage.measure import compare\_ssim

import argparse

import imutils

import cv2

# load the two input images

imageA = cv2.imread(args["first"])

imageB = cv2.imread(args["second"])

# convert the images to grayscale

grayA = cv2.cvtColor(imageA, cv2.COLOR\_BGR2GRAY)

grayB = cv2.cvtColor(imageB, cv2.COLOR\_BGR2GRAY)

# compute the Structural Similarity Index (SSIM) between the two

# images, ensuring that the difference image is returned

(score, diff) = compare\_ssim(grayA, grayB, full=True)

diff = (diff \* 255).astype("uint8")

print("SSIM: {}".format(score))

# threshold the difference image, followed by finding contours to

# obtain the regions of the two input images that differ

thresh = cv2.threshold(diff, 0, 255, cv2.THRESH\_BINARY\_INV | cv2.THRESH\_OTSU)[1]

cnts = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

# loop over the contours

for c in cnts:

# compute the bounding box of the contour and then draw the

# bounding box on both input images to represent where the two

# images differ

(x, y, w, h) = cv2.boundingRect(c)

cv2.rectangle(imageA, (x, y), (x + w, y + h), (0, 0, 255), 2)

cv2.rectangle(imageB, (x, y), (x + w, y + h), (0, 0, 255), 2)

# show the output images

cv2.imshow("Original", imageA)

cv2.imshow("Modified", imageB)

cv2.imshow("Diff", diff)

cv2.imshow("Thresh", thresh)

cv2.waitKey(0)

**4.4 IMPLEMENTATION:**

Using this method, we were able to easily determine if two images were identical or had differences due to slight image manipulations, compression artifacts, or purposeful tampering.

**Structural similarity index measure:**

The structural similarity index measure (SSIM) is a method for predicting the perceived quality of digital television and cinematic pictures, as well as other kinds of digital images and videos. SSIM is used for measuring the similarity between two images. The SSIM index is a full reference metric; in other words, the measurement or prediction of image quality is based on an initial uncompressed or distortion-free image as reference.

SSIM is a perception-based model that considers image degradation as perceived change in structural information, while also incorporating important perceptual phenomena, including both luminance masking and contrast masking terms. The difference with other techniques such as MSE or PSNR is that these approaches estimate absolute errors. Structural information is the idea that the pixels have strong inter-dependencies especially when they are spatially close. These dependencies carry important information about the structure of the objects in the visual scene. Luminance masking is a phenomenon whereby image distortions (in this context) tend to be less visible in bright regions, while contrast masking is a phenomenon whereby distortions become less visible where there is significant activity or "texture" in the image.

**4.5 SOFTWARE TOOLS**

To accomplish this, we’ll first need to make sure our system has Python, OpenCV, scikit-image, and imutils

**1) OpenCV Python:**

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as NumPy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in NumPy can be combined with OpenCV.

This OpenCV tutorial will help you learn the Image-processing from Basics to Advance, like operations on Images, Videos using a huge set of OpenCV-programs and projects.

Instead software tools

**2) Scikit- image:**

scikit-image (formerly scikits.image) is an open-source image processing library for the Python programming language.[2] It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more.[3] It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

**Important features of Scikit-image:**

Simple and efficient tools for image processing and computer vision techniques.

Accessible to everybody and reusable in various contexts.

Built on the top of NumPy, SciPy, and matplotlib.

Open source, commercially usable – BSD license.

**3) Imutils:**

A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much easier with OpenCV and both Python 2.7 and Python 3.

Chapter 5: TESTING

5.1 SOFTWARE TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation.

## 5.1.1 Testing Objectives

* + To ensure that during operation the system will perform as per specification.
  + TO make sure that system meets the user requirements during operation
  + To make sure that during the operation, incorrect input, processing and output will be detected
  + To see that when correct inputs are fed to the system the outputs are correct
  + To verify that the controls incorporated in the same system as intended
  + Testing is a process of executing a program with the intent of finding an error
  + A good test case is one that has a high probability of finding an as yet undiscovered error.

The software developed has been tested successfully using the following testing strategies and any errors that are encountered are corrected and again the part of the program or the procedure or function is put to testing until all the errors are removed. A successful test is one that uncovers an as yet undiscovered error.

Note that the result of the system testing will prove that the system is working correctly. It will give confidence to system designer, users of the system, prevent frustration during implementation process etc.,

## 5.2 TEST CASE DESIGN:

## 5.2.1 White box testing

White box testing is a testing case design method that uses the control structure of the procedure design to derive test cases. All independents path in a module are exercised at least once, all logical decisions are exercised at once, execute all loops at boundaries and within their operational bounds exercise internal data structure to ensure their validity. Here the customer is given three chances to enter a valid choice out of the given menu. After which the control exits the current menu.

## 5.2.2 Black Box Testing

Black Box Testing attempts to find errors in following areas or categories, incorrect or missing functions, interface error, errors in data structures, performance error and initialization and termination error. Here all the input data must match the data type to become a valid entry.

The following are the different tests at various levels:

**5.2.3 Unit Testing:**

Unit testing is essentially for the verification of the code produced during the coding phase and the goal is test the internal logic of the module/program. In the Generic code project, the unit testing is done during coding phase of data entry forms whether the functions are working properly or not. In this phase all the drivers are tested they are rightly connected or not.

**5.2.4 Integration Testing:**

All the tested modules are combined into sub systems, which are then tested. The goal is to see if the modules are properly integrated, and the emphasis being on the testing interfaces between the modules. In the generic code integration testing is done mainly on table creation module and insertion module.

## 5.2.5 Validation Testing

This testing concentrates on confirming that the software is error-free in all respects. All the specified validations are verified and the software is subjected to hard-core testing. It also aims at determining the degree of deviation that exists in the software designed from the specification; they are listed out and are corrected.

## 5.2.6 System Testing

This testing is a series of different tests whose primary is to fully exercise the computer-based system. This involves:

* Implementing the system in a simulated production environment and testing it.
* Introducing errors and testing for error handling.

**5.3 TEST CASES**

**5.3.1 Test Case 1:**

**Input type:**

The images that were given as input should be of .png or .jpg format.

**5.3.2 Test Case 2:**

**Image dimensions:**

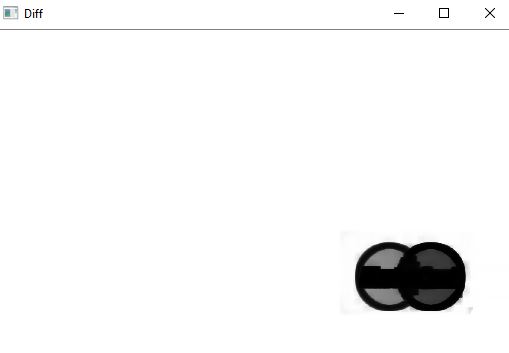
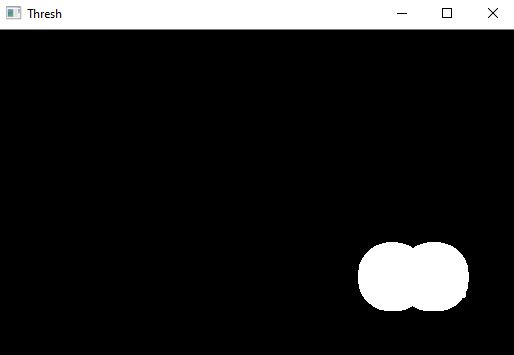
Both the images given as input must have same dimensions.

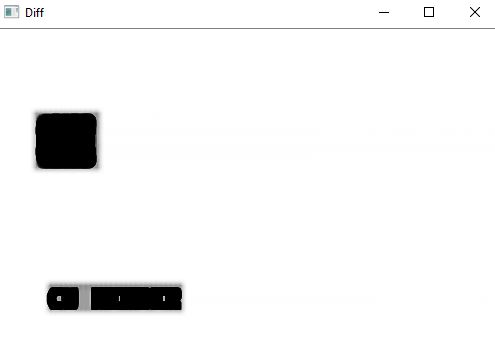
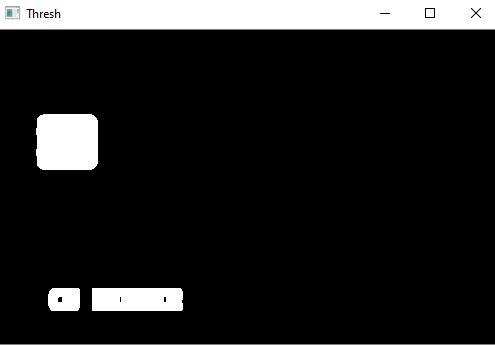
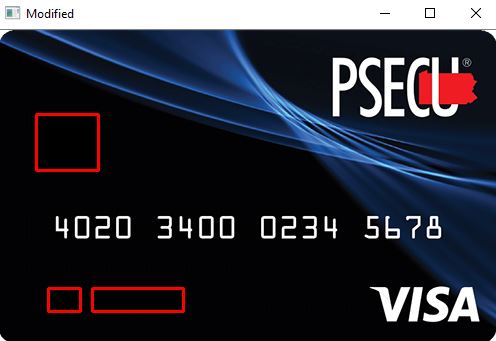
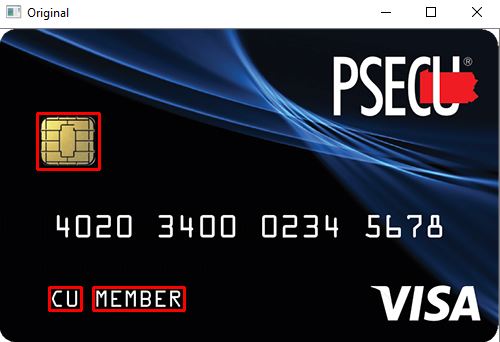
**5.3.3 Test Case 3:**

**Input:**

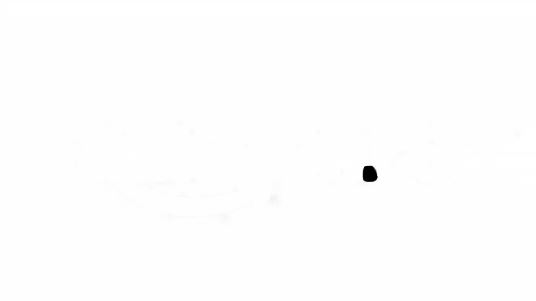
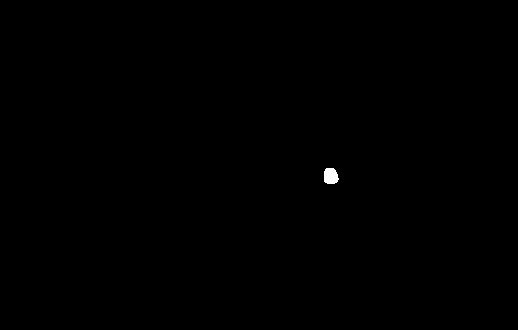
Only images are taken as input videos are not accepted.

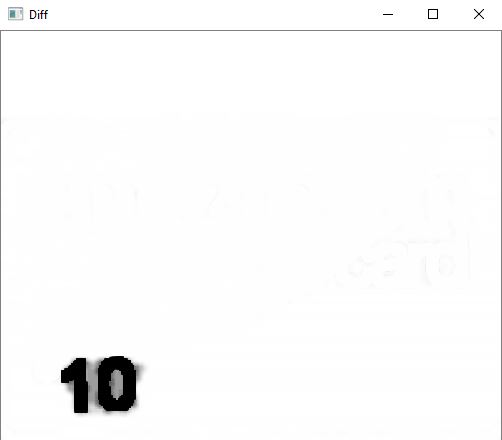
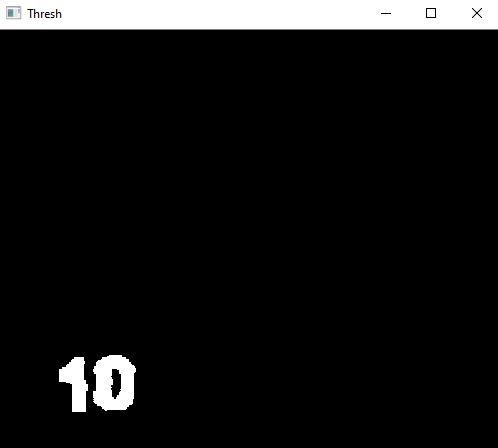
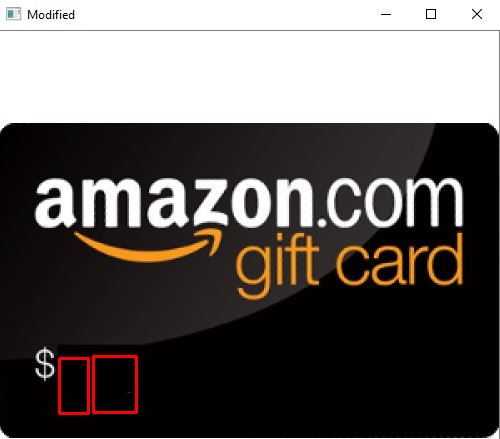
**Chapter 6:** SCREEN SHOTS

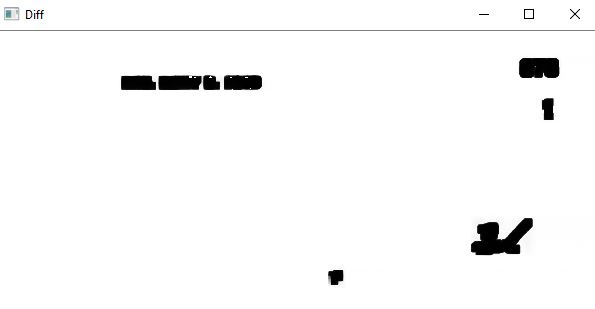
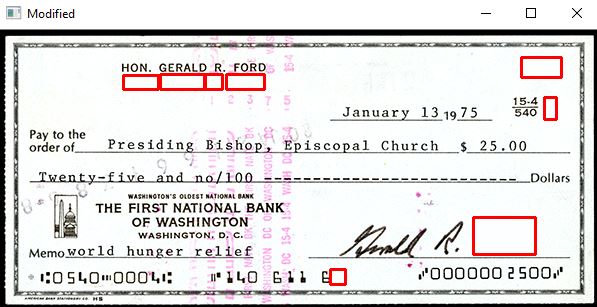
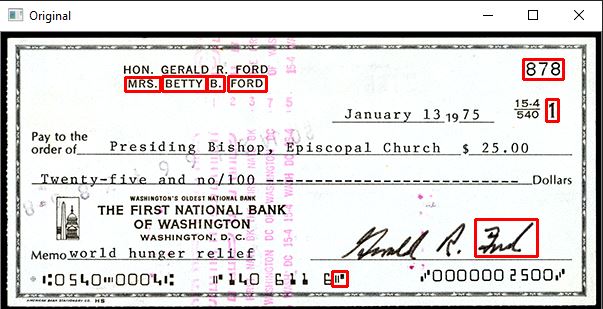


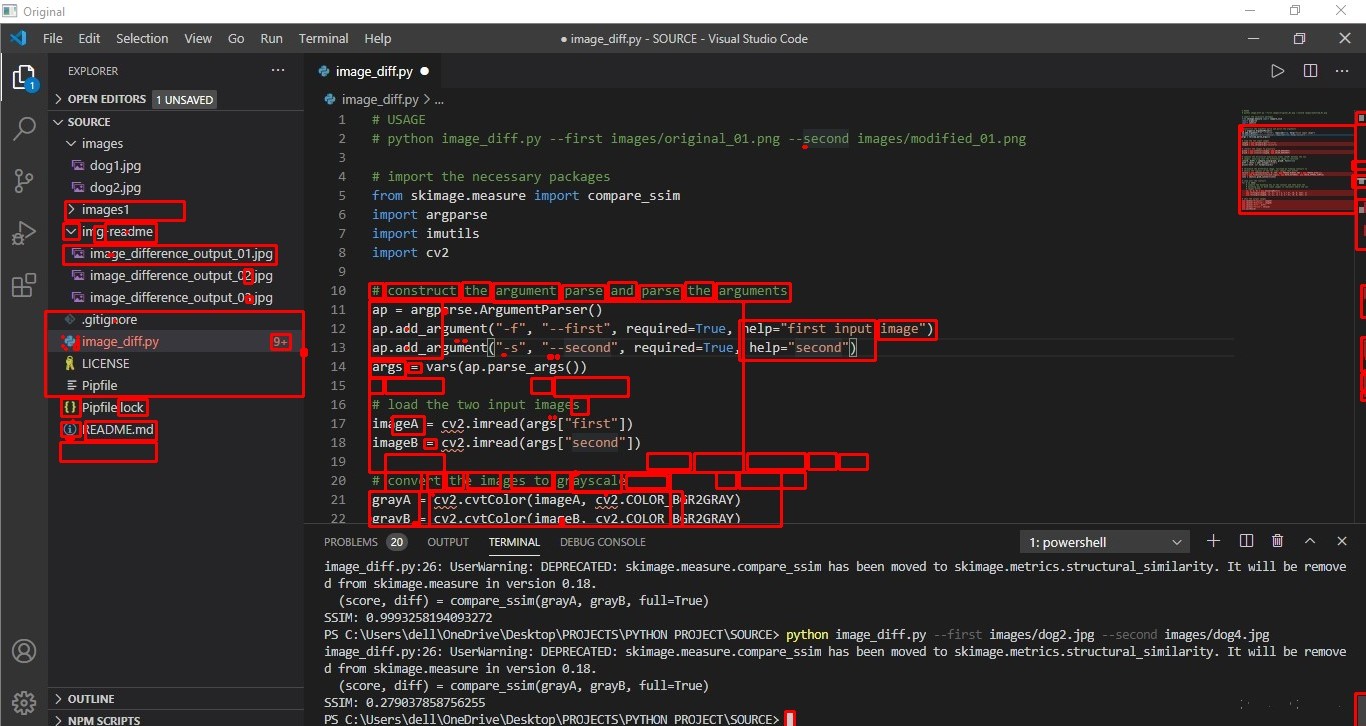


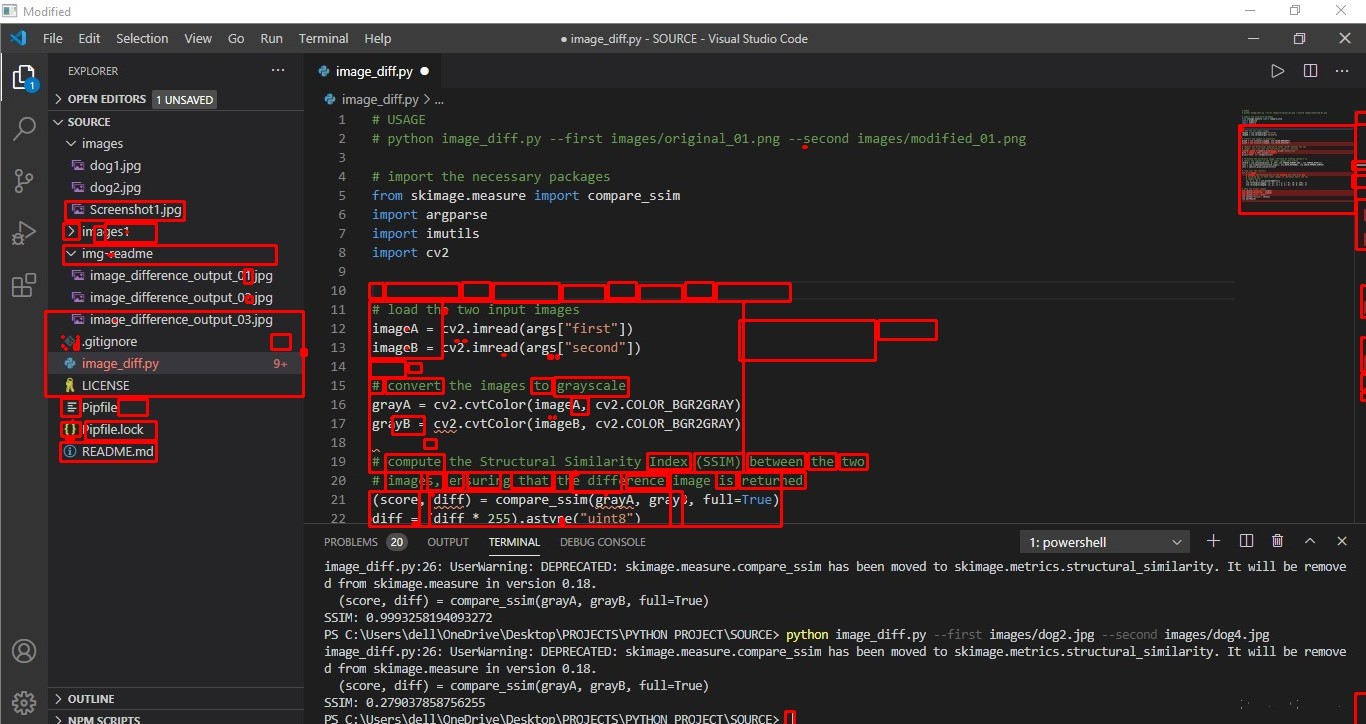


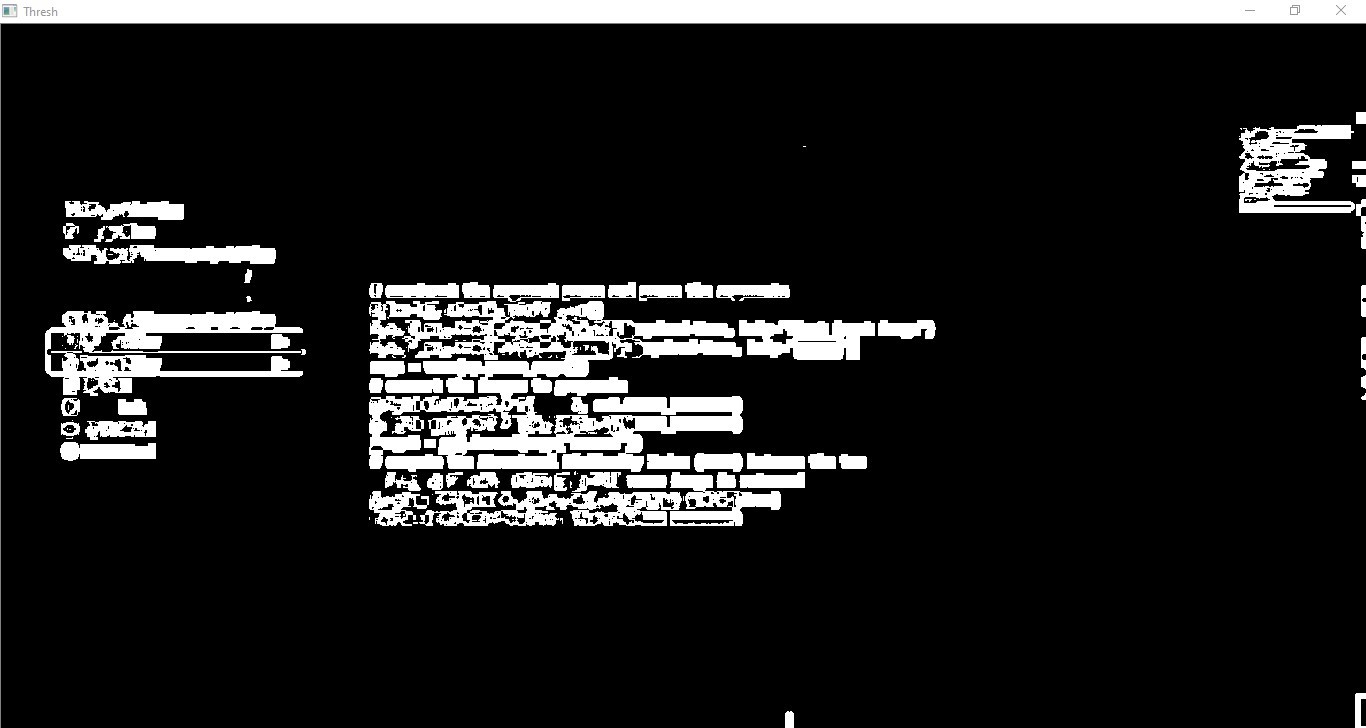














Chapter 7: CONCLUSION

This proposed a real time image difference recognition system. The image difference recognition is implemented using python and industrial standard tool Open Computer Vision. The experiment consists of applying two images and recognize if two images were identical or had differences due to slight image manipulations, compression artifacts, or purposeful tampering. An important aspect of comparing image differences is sometimes image differences are so subtle that the naked eye struggles to immediately comprehend the difference. The practical results even demonstrated the slight differences between two images. Finally, this proposed system could help identify the manipulated images and by applying these techniques we can also develop a phishing detection system.

Chapter 8: BIBILOGRAPHY

Books used for collecting the information are:

1. The Complete Reference Python by Martin C.Brown, published by Osborne/McGraw-Hill, 2001.
2. Computer Vision algorithms and applications by Richard Szeliski.
3. Unified Modeling Language User Guide (2nd edition) by Grady Booch, James Rumbaugh and Ivar Jacobson published by Pearson Education.
4. Software Engineering - Practitioner’s Approach (6th edition) by Rodger S. Pressman, published by McGraw- Hill Education in the year 2004.

The following websites are used for collecting the information:

[www.google.com](http://www.google.com)

<http://www.python.org/>

<http://opencv.org/>

<http://scikit-image.org/>

<http://stackoverflow.com/>