**TrueSight: Image Authentication**

**A project report**

**submitted in partial fulfilment of the**

**requirements for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE & ENGINEERING**

**Submitted by**

**SHIKHA SINGH** (2107510100048)

**SHAVEZ KHAN** (2107510100048)

**CHANDAN PANDEY** (2107510100018)

**ABHISHEK SINGH** (2107510100005)

**SANDHYA SINGH** (2107510100045)

**Under the Guidance of**

**ER. VIVEK PATEL**



To the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

KIPM COLLEGE OF ENGINEERING AND TECHNOLOGY, GORAKHPUR

**Dr APJ Abdul Kalam Technical University Lucknow**

**May, 2025**

**TrueSight: Image Authentication**

**A project report**

**submitted in partial fulfilment of the**

**requirements for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE & ENGINEERING**

**Submitted by**

**SHIKHA SINGH** (2107510100048)

**SHAVEZ KHAN** (2107510100048)

**CHANDAN PANDEY** (2107510100018)

**ABHISHEK SINGH** (2107510100005)

**SANDHYA SINGH** (2107510100045)

**Under the Guidance of**

**ER. VIVEK PATEL**



To the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

KIPM COLLEGE OF ENGINEERING AND TECHNOLOGY, GORAKHPUR

**Dr APJ Abdul Kalam Technical University Lucknow**

**May, 2025**

# CERTIFICATE

This is to certify that the project work entitled **“TrueSight: An Image Recognition System”** is a bona fide work carried out by **Shikha Singh, Shavez Khan, Sandhya Sahani, Chandan Pandey, and Abhishek Singh**. This project has been submitted in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science & Engineering** under **Dr. A.P.J. Abdul Kalam Technical University**, for the academic session **2024–2025**.

It is certified that all corrections and suggestions recommended for internal assessment have been duly incorporated in the final report. The project embodies the results of original work conducted by the students and does not contain any material previously published or written by another person, nor material that has been accepted for the award of any other degree or diploma of any other university or institution, except where due acknowledgment has been made.

The project work has been found satisfactory and approved for submission.

SHIKHA SINGH (2107510100049)

SHAVEZ KHAN (2107510100048)

CHANDAN PANDEY (2007510100018)

SANDHYA SAHANI (2007510100034)

ABISHEK SINGH (2107510100005)

Er. Vivek Patel Er. Ranjeet Kumar Dubey

**Project Guide Project Coordinator**

Dr. Ranjeet Rai

**Head of Department - CSE** **External Examiner**

# DECLARATION

We hereby declare that this submission is our own work as requirements of major project and that are the best of our knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

**Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Name:** Shikha Singh **Name:** Chandan Pandey

**Roll No.:** 2107510100049 **Roll No.:** 2007510100018

**Date:** / /2025 **Date:** / /2025

**Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Name:** Shavez Khan **Name:** Sandhya Sahani

**Roll No.:** 2107510100048 **Roll No.:** 2107510100045

**Date:** / /2025 **Date:** / /2025

**Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Name:** Abhishek Singh

**Roll No.:** 2107510100005

**Date:** / /2025

# ACKNOWLEDGEMENT

With deep gratitude I express my earnest thanks to my esteemed guide Mr. Ranjeet Kumar Dubey (Assistant Professor), Department of Computer Science & Engineering for his constant involvement, energetic efforts and proficient guidance, which gave me direction and body to work, reported here. Without his wise counsel and encouragement, it would have been impossible to complete the project work in this manner.

I am thankful to all the faculty of the Computer Science & Engineering Department especially for their intellectual support during my project work.

I also want to thank to my friends for their valuable support whenever I needed. I would like to thank all those people who have helped me some way or the other in my thesis work.

Lastly, and most importantly, I thank my parents for their moral support and encouragement towards completing my thesis successfully. In the last, I want to thank Almighty God.

Date: Shikha Singh (2107510100049)

Shavez Khan (2107510100048)

Abhishek Singh (2107510100005)

Chandan Pandey (2107510100034)

Sandhya Sahani (2107510100045)

# ABSTRACT

TrueSight is a Python-based image recognition system developed to classify and analyse images using modern machine learning techniques. This project presents a comprehensive solution for automated image identification, focusing on key aspects such as model training, visual interpretation, and real-time prediction. The system employs deep learning algorithms using TensorFlow, supported by Seaborn and Matplotlib for data visualization and analysis throughout the development process.

The core of the project is built around convolutional neural networks (CNNs), which have been trained on labelled datasets to enable accurate object recognition and classification. The model evaluates input images, processes pixel data, and delivers predicted outcomes with associated confidence scores. An interactive interface is designed to allow users to upload images, view real-time predictions, and understand model behaviour through plotted graphs and heatmaps.

Data visualization plays a vital role in this project. Seaborn and Matplotlib are used to generate plots such as training/validation accuracy curves, confusion matrices, and class-wise performance indicators. These insights help validate the robustness of the model and highlight areas for improvement. The system is structured to be extendable, supporting future enhancements such as image segmentation and multi-class prediction.

TrueSight aims to make intelligent image classification accessible for applications in sectors like education, agriculture, security, and healthcare. With a focus on clarity, usability, and performance, the project integrates essential tools and methodologies to support visual understanding in a computationally efficient manner. By combining Python’s scientific stack with machine learning, TrueSight showcases the potential of AI-powered vision systems in modern software development**.**

# TABLE OF CONTENTS

CERTIFICATE I

DECLARATION ii

ACKNOWLEDGEMENT iii

ABSTRACT iv

TABLE OF CONTENTS v

LIST OF FIGURES VI

CHAPTER 1: INTRODUCTION 1

1.1 Context: 2

1.2 Problem statement: 5

1.3 Scope of the Project: 7

1.4 Proposed solution: 9

CHAPTER 2: literature survey 11

2.1 Existing Deepfake Detection techniques 12

2.2 Limitations of the Existing Methods 16

2.3 Research Gap Edentified 18

CHAPTER 3: methodology 19

3.1 Dataset Description 20

3.2 Data Processing Techniques 22

3.3 Model Architecture 25

3.4 Summary of CNN Layers Workflow 30

3.5 TPU Compatibility 31

3.6 Training Strategy(Callbsck, Early Stopping, ETC.) 33

CHAPTER 4: TOOLS AND REQUIReMENTS 36

4.1) Development Environment 37

4.2) Libraries And Frameworks 39

4.3) Model TRaining 42

4.4) Model Evaluation 45

4.5) Visualization: Accuracy And Loss Graphs 46

CHAPTER 5: CODE SNAPSHOTS AND OUTPUT 47

5.1 Flask Framework Overview 48

5.2 Front-End Design 52

5.3 Backend Logic for Model Integration 72

5.4 Image Upload And Integration Flow 72

5.5 Local Website TEsting 73

5.6 Output ScreenShots 74

CHAPTER 6: Results and analysis 78

5.2 Front-End Design 79

5.3 Backend Logic for Model Integration 79

5.4 Image Upload And Integration Flow 79

5.5 Local Website TEsting 80

5.6 Output ScreenShots 80

# LIST OF FIGURES

**Figure 1.1** GDP Partition……………………………………………………………………….2

**Figure 2.1** MERN Stack Development………………………………………………………..16

**Figure 3.1.1** Apani Fashal Apani Suraksha Use Case Diagram …….……………………….....18

**Figure 3.1.2** Dukan Use Case Diagram…….…………………………………………………...19

**Figure 3.2.1** Dukan ER Diagram……………………………………………………………..…20

**Figure 3.2.2** Apani Fashal Apani Suraksha ER Diagram…………………………………….....21

**Figure 3.3.1** DFD Dukan……………………………………………………………….……22-23

**Figure 3.3.2** DFD Apani Fashal Apani Suraksha……………………………………………24-25

**Figure 4.3.1** Apani Fashal Apni Suraksha Project Folder Structure……………………………..28

**Figure 4.3.2** Dukaan Project Folder Structure……………………………………………………28

**Apani Fashal Apani Suraksha**

**Figure 5.2.1** Register Page…………………………………………………………………… 74

**Figure 5.2.2** Login Page………………………………………………………………………...74

**Figure 5.2.3** Forgot Password Page……………………………………………………………..75

**Figure 5.2.4** Home Page………………………………………………………………………...75

**Figure 5.2.5** News Page…………………………………………………………………………76

**Figure 5.2.6** Module Page……………………………………………………………………….76

**Figure 5.2.7** Community Page…………………………………………………………………..77

**Dukaan**

**Figure 5.3.1** Login Page………………………………………………………………………...77

**Figure 5.3.2** Register Page………………………………………………………………………78

**Figure 5.3.3** Forgot Password Page……………………………………………………………..78

**Figure 5.3.4** Home Page……………………………………………………………………….79

**Figure 5.3.5** Product Details Page……………………………………………………………..79

**Figure 5.3.6** Cart Page…………………………………………………………………………80

**Figure 5.3.7** Checkout Page……………………………………………………………………80

**Figure 5.3.8** Payment Page…………………………………………………………………….81

**Figure 5.3.9** User Order History Page…………………………………………………………81

**Figure 5.3.10** Admin Dashboard Page…………………………………………………………..82

**Figure 5.3.11** Seller Dashboard Page…………………………………………………………...82

**Figure 5.3.12** Team Page………………………………………………………………………..83

# CHAPTER 1

# INTRODUCTION

# CHAPTER 1

# INTRODUCTION

## Context:

The ubiquity of digital images in modern society necessitates reliable methods for verifying their authenticity. With the ease of digital manipulation and the emergence of sophisticated image generation technologies, distinguishing between genuine and synthetic visuals has become increasingly challenging. This raises critical concerns across various domains, where the veracity of image content is paramount.

This project introduces "TrueSight," an image recognition system developed to analyse input images and determine whether they are likely to be real or not. TrueSight employs advanced image processing techniques and potentially machine learning models to scrutinize image characteristics and identify patterns or anomalies that may indicate manipulation or artificial generation.

The primary goal of TrueSight is to provide a system capable of evaluating the authenticity of digital images. By examining visual cues, statistical properties, and potential artifacts introduced through tampering or synthesis, TrueSight aims to offer a determination of an image's likelihood of being genuine.

The ability to ascertain the authenticity of an image has significant implications across numerous fields. In news and media, it can aid in verifying the integrity of visual evidence. In security and forensics, it can be crucial for validating the authenticity of photographic records. Moreover, in personal and professional communication, it can help individuals assess the trustworthiness of shared visual content.

In the broader context of digital trust and information integrity, TrueSight addresses the growing need for tools that can help users navigate an environment where visual misinformation can spread rapidly. This project will focus on the development and evaluation of TrueSight's capabilities in analyzing images and providing a classification of whether they are real or not, contributing to the ongoing efforts to ensure the reliability of digital visual information.

The digital image has become an indispensable medium for communication, documentation, and information dissemination across all facets of modern life. From personal sharing on social media to critical applications in journalism, science, and law enforcement, images play a vital role. However, this widespread reliance on digital imagery is increasingly challenged by the growing sophistication and accessibility of image manipulation and generation technologies.

The ability to seamlessly alter existing images or create entirely synthetic ones ("deepfakes" or AI-generated images) has created a significant problem. These manipulated or fabricated images can be used to spread misinformation, deceive individuals, damage reputations, commit fraud, or undermine trust in visual evidence. The potential for harm necessitates the development of robust and reliable methods for verifying the authenticity of digital images.

This project introduces "TrueSight," an advanced image recognition system designed to address this critical need. TrueSight goes beyond simple detection of obvious forgeries. It employs a multi-faceted approach, integrating advanced image processing, feature extraction, and potentially deep learning techniques, to analyse images at a granular level. The system is designed to:

* **Analyze Pixel-Level Integrity:** TrueSight examines the fundamental structure of an image, looking for inconsistencies in pixel relationships, statistical anomalies, and signs of tampering that might not be visible to the naked eye. This includes detecting traces of resampling, cloning, or other manipulation techniques.
* **Evaluate Metadata Consistency:** TrueSight analyzes image metadata (information embedded within the image file) for discrepancies or inconsistencies that may indicate manipulation. While metadata can be altered, its analysis provides valuable clues.
* **Identify Artificial Artifacts:** With the rise of AI-generated images, TrueSight is designed to detect artifacts or patterns that are characteristic of specific generation algorithms. This includes identifying unique noise patterns, color distributions, or structural inconsistencies that differentiate synthetic images from real-world photographs.
* **Learn and Adapt:** TrueSight, depending on its architecture, can potentially leverage machine learning to continuously improve its accuracy and adapt to emerging manipulation and generation techniques. By training on large datasets of both authentic and manipulated images, the system can learn to recognize subtle indicators of forgery.

The applications of TrueSight are extensive and impactful:

* **Combating Misinformation:** TrueSight can be deployed on social media platforms and news websites to help identify and flag potentially manipulated images, reducing the spread of false information.
* **Strengthening Legal Proceedings:** In courtrooms and forensic investigations, TrueSight can provide crucial evidence regarding the authenticity of photographic evidence, ensuring the integrity of the judicial process.
* **Protecting Intellectual Property:** TrueSight can help verify the authenticity of images used in advertising, art, and other creative industries, preventing copyright infringement and fraud.
* **Enhancing Security:** TrueSight can be used in security systems to verify the authenticity of images captured by surveillance cameras or other imaging devices.

In the broader context of the information age, TrueSight contributes to building a more trustworthy digital ecosystem. By providing a sophisticated tool for image authenticity verification, this project

addresses the urgent challenge of maintaining the integrity of visual information in a world where manipulation is becoming increasingly prevalent. The project will involve the detailed exploration

of algorithms, model development, performance evaluation, and potential limitations of TrueSight, ultimately advancing the field of image forensics and digital trust.

## Problem statement:

Despite rapid advancements in artificial intelligence and computer vision, many sectors still face significant challenges when it comes to accurately analyzing and interpreting visual data. Manual image classification is time-consuming, error-prone, and often impractical for large datasets or real-time scenarios. Industries like agriculture, security, healthcare, and manufacturing require efficient systems that can detect and recognize objects in images with minimal human intervention.

One major challenge is the lack of accessible and user-friendly platforms that provide high-accuracy image recognition without the need for extensive technical expertise. Many existing systems require complex setups, large-scale infrastructure, or cloud-based processing, which may not be feasible for small-scale or offline applications. In addition, the interpretation of image recognition results is often not visualized clearly, making it difficult for end users to understand model behavior or trust predictions.

Training and deploying deep learning models for image recognition involves several technical barriers, including dataset preparation, model tuning, and performance evaluation. For students, researchers, or domain experts without a strong machine learning background, this limits the ability to experiment and adopt these technologies.

There is also a need for localized, interpretable, and lightweight image recognition tools that are suitable for educational and research settings. These tools should not only classify images accurately but also provide visual insights like confusion matrices, training curves, and class distributions to help users understand model performance and limitations

TrueSight aims to solve these problems by offering an integrated Python-based image recognition system. It uses TensorFlow to power its neural network model and Seaborn and Matplotlib to generate detailed visualizations. This project targets the creation of a lightweight, offline-capable tool that delivers accurate image classification and insightful performance metrics in a simple, accessible format.

**1. Inaccessibility of Image Recognition Tools**

Many image recognition systems are complex, resource-intensive, and require specialized knowledge for setup and use. This creates a barrier for students, educators, and small organizations who may need a simplified yet effective solution.

**2. Dependence on Manual Image Classification**

Manual sorting and analysis of image data is time-consuming, inefficient, and error-prone, especially in scenarios that demand speed and scale such as surveillance, medical diagnostics, or agricultural monitoring.

**3. Lack of Model Interpretability**

Most existing image classification tools do not provide adequate insights into how predictions are made. Without visual feedback like confusion matrices or training curves, users struggle to understand and trust the model's outcomes.

**4. Technical Complexity in Model Deployment**

Building and deploying deep learning models typically requires a strong understanding of Python, neural networks, and data preprocessing. This steep learning curve discourages adoption among non-experts.

**5. Need for Lightweight, Offline Solutions**

Current solutions often rely on cloud-based systems and large infrastructure, which limits use in bandwidth-constrained or offline environments, particularly in rural or field-based applications.

**TrueSight** aims to overcome these challenges by providing a Python-based, lightweight image recognition tool that uses **TensorFlow** for deep learning and **Matplotlib/Seaborn** for performance visualization. It simplifies user interaction, offers real-time feedback, and promotes understanding through clear visual outputs, making it accessible for educational, research, and real-world deployment.

## Scope of the Project:

**TrueSight** is designed to recognize and classify **human face images** as **real or fake**, with a focus on detecting AI-generated or manipulated faces. The project uses machine learning techniques to help users identify whether a human image is genuine or artificially generated using deepfake or GAN-based tools.

#### ****1. Detecting AI-Generated Faces****

The system can analyze a given human face image and determine if it is a real photograph or a computer-generated fake. This helps in identifying deepfakes or synthetic media.

#### ****2. Use in Social Media and News****

TrueSight can help reduce the spread of misinformation by detecting fake human images shared on social media platforms or in fake news content.

#### ****3. Educational Purpose****

The project helps students and researchers understand how deepfake detection works, how image patterns differ between real and fake faces, and how machine learning models can learn to recognize them.

#### ****4. Security and Verification****

TrueSight can be useful in security systems to verify if a human image is authentic. It can also help prevent identity fraud using fake profile pictures.

#### ****5. Support for Digital Forensics****

It may assist forensic experts and investigators in analyzing visual content for signs of digital tampering, which is useful in cybercrime investigations.

#### ****6. Offline Usage****

TrueSight works without needing an internet connection. This makes it helpful for use in secure environments or areas with low connectivity.

#### ****7. Lightweight and Accessible****

The system runs using Python and does not require high-end hardware. It is suitable for use on standard systems, making it accessible for schools, small labs, and individuals.

#### ****8. Visual Feedback****

Users can see visual graphs and results to better understand how the system is classifying the images. This includes accuracy scores, predictions, and sample outputs.

#### ****9. Base for Future Enhancements****

The current system focuses on image classification, but it can be expanded in the future to detect fake videos, improve model accuracy, or analyze face features in more detail.

#### ****10. Promotes Ethical Use of AI****

By helping detect fake human faces, TrueSight contributes to the ethical and responsible use of AI, protecting users from manipulation and fraud.

## Proposed solution:

#### ****1. Upload and Recognize****

Users can upload any image into the system. The system looks at the image and tells what object or thing is present in it.

#### ****2. Trained Model****

We trained the system using many pictures so it could learn how different objects look. This helps the system make correct guesses for new images.

#### ****3. Easy Interface****

The system is designed to be simple. Anyone can use it without needing to know coding or technical things.

#### ****4. Confidence Score****

After guessing the object in the image, the system shows how confident it is (e.g., 95% sure it’s a cat).

#### ****5. Graphs and Charts****

The system also shows graphs like accuracy and confusion matrix, so users can understand how well it is performing.

#### ****6. Offline Use****

TrueSight works without the internet. This makes it useful in villages, schools, or places with no network.

#### ****7. Fast and Lightweight****

The system runs fast and does not need high-end computers. It works well on normal systems too.

#### ****8. Multiple Uses****

It can be used in areas like education, agriculture, security, and health to quickly recognize what is in an image.

#### ****9. No Extra Setup****

There is no need for online servers or databases. It works using simple Python code and tools like TensorFlow, Seaborn, and Matplotlib.

#### ****10. Future Ready****

We can improve it further in future by adding features like live camera input, support for more objects, or recognizing many objects in one image.

# CHAPTER 2

# LITERATURE SURVEY

# CHAPTER 2

# LITERATURE SURVEY

# EXISTING DEEPFAKE DETECTION TECHNIQUES

With the rapid development of Generative Adversarial Networks (GANs) and face manipulation tools, the creation of realistic but fake human faces has become easy and widespread. Detecting such manipulations is now a serious concern for privacy, security, and digital trust. Below are the major techniques developed to detect deepfakes and synthetic human images:

**1. Deep Learning-Based Approaches**

These are currently the most widely used methods. Deep learning models, especially Convolutional Neural Networks (CNNs), are trained on large datasets containing both real and fake faces.

How it works: CNNs learn to identify subtle patterns that differentiate real and fake images, such as unnatural blending, artifacts, and distortions. These features may not be visible to the human eye but are captured by the model during training.

**Popular models:**

* + **XceptionNet**: Known for high performance in the FaceForensics++ dataset.
  + **ResNet**: Deep residual learning for feature extraction.
  + **EfficientNet**: Optimized for performance on low-resource systems.
* Strengths: High accuracy, adaptable to different datasets.
* Limitations: Requires large labelled datasets and high computational power during training.

**2. Frequency-Based Analysis**

GANs often introduce high-frequency artifacts due to up sampling layers (like transpose convolutions). These artifacts are invisible in spatial domains but visible in frequency domains.

* How it works: Image is transformed using Fourier Transform (FFT/DFT) or Wavelet Transform to analyse its frequency components. Real and fake images show different frequency patterns.
* **Models & Tools:**
  + F3Net: Focuses on frequency-aware features in deepfake detection.
  + Spectrum-based CNNs: Analyze DFT-transformed image spectra.
* Strengths: Useful for detecting GAN-specific signatures.
* Limitations: May be bypassed by advanced GANs that mimic natural frequency spectra.

**3. Physiological Signal Analysis**

Human faces in real videos or images reflect natural biological rhythms and behaviours, which are difficult for deepfakes to replicate accurately.

* **Techniques used:**
  + Eye Blinking Detection: Deepfakes often miss natural blinking frequency or synchronization.
  + Heart Rate Detection: Using tiny colour changes in skin (Photoplethysmography).
  + Lip-Sync Analysis: Detecting mismatches between spoken audio and mouth movement.
* Strengths: Hard to replicate physiological patterns in generated content.
* Limitations: Mostly applicable to videos, not single images.

**4. Artifact Detection**

Fake images often contain invisible artifacts due to GAN architecture flaws, such as:

* Irregular reflections in the eyes
* Incorrect ear symmetry
* Mismatched lighting and shadow directions
* Blurry borders near facial features

These inconsistencies are detected using specialized CNN models or attention mechanisms that focus on local pixel regions.

* Examples:
  + Face X-ray: Detects blending artifacts.
  + Patch-based CNNs: Divide face into segments for focused analysis.
* Strengths: Works well on high-resolution images.
* Limitations: Advanced GANs may reduce these flaws significantly.

**6. Metadata and Compression Trace Analysis**

Every digital image or video contains metadata, including camera details, timestamps, GPS, and file compression traces.

* **How it helps:**
  + Fake images might have missing or manipulated metadata.
  + GAN-generated images usually lack natural camera sensor noise.
* **Tools**:
  + EXIF analysis software
  + Photo Forensics tools
* Strengths: Quick checks without analyzing visual content.
* Limitations: Can be easily removed or faked.

**7. Hybrid and Ensemble Models**

Combining different detection techniques can improve overall accuracy.

* A hybrid system might include:
  + CNN for spatial features
  + Frequency analyzer for artifacts
  + Metadata checker for file-level validation

Such systems are more robust and can detect a wider variety of deepfake strategies.

* Example: Top entries in Facebook’s Deepfake Detection Challenge (DFDC) used hybrid models with spatial, frequency, and audio features.

**8. Pretrained Models and Public APIs**

Several research organizations and companies have released tools and APIs for detecting fake media:

* **Microsoft Video Authenticator**: Detects manipulated facial content.
* **Sensity AI**: Cloud-based detection for social media platforms.
* **Deepware Scanner**: Web-based tool to identify deepfake content.

**Conclusion**

Detecting deepfakes is a constantly evolving challenge. While many deep learning and signal-based techniques show great promise, fake image generators are also improving in quality. Therefore, ongoing research, dataset updates, and hybrid model development are essential to stay ahead of fake content creators.

In the context of TrueSight, these techniques provide a foundation and comparison point for building a lightweight, image-based deepfake detection system tailored for detecting fake human faces.

1. **LIMITATIONS OF EXISTING METHODS**

**1. Generalization to Unseen Data**

Most deepfake detection models are trained on specific datasets and often struggle to perform well when tested on unseen GANs or manipulation techniques. As new deepfake generators emerge, existing models may fail to detect unfamiliar patterns, leading to reduced accuracy.

**2. Overfitting to Training Artifacts**

Many detection models unintentionally learn dataset-specific noise or artifacts rather than generalizable features. This results in high performance on known datasets but poor real-world applicability, where deepfakes may have different characteristics.

**3. Lack of Robustness Against Post-Processing**

Simple image or video post-processing techniques like compression, resizing, rotation, or adding noise can significantly reduce detection accuracy. Many existing systems are sensitive to such common transformations, making them unreliable in practical settings.

**4. Limited Interpretability**

Deep learning-based detection models often act as black boxes, providing little to no insight into why a prediction was made. This lack of transparency reduces trust in the model and makes it difficult for users to validate results or spot errors.

**5. Inapplicability to Still Images (in some methods)**

Several techniques, especially those based on temporal inconsistencies or physiological signals (e.g., eye blinking or lip-sync), are designed for videos and do not work on single image inputs—limiting their use in detecting fake profile pictures or AI-generated faces.

**6. High Computational Requirements**

State-of-the-art deepfake detection models often require significant GPU resources and memory for both training and inference. This makes them unsuitable for deployment on low-resource devices or in offline environments.

**7. Privacy and Ethical Concerns**

Some detection tools may require uploading media to external servers (e.g., cloud APIs), raising concerns about privacy, data security, and misuse—especially when analyzing sensitive images or videos.

**8. Fragile Against Adversarial Attacks**

Deepfake detectors can be fooled by adversarial examples, where slight, imperceptible changes are made to images or videos to bypass detection. Attackers can exploit these weaknesses to make fake content appear legitimate.

**9. Dataset Bias**

Most public datasets used for training are limited in terms of ethnicity, gender, lighting conditions, and camera types. This leads to biased performance, where the model might work better on certain faces and poorly on others.

**10. Real-Time Detection Limitations**

While many models achieve good accuracy, very few are optimized for real-time performance. High processing time and model size make them unsuitable for live applications like surveillance systems or mobile use.

1. **Research Gap Identified**

Despite the rapid advancements in deepfake detection using AI and machine learning techniques, several research gaps still remain:

1. **Low Accuracy on Real-World Data:**  
   Many existing models perform well on benchmark datasets but fail to maintain the same accuracy when tested on real-world manipulated images due to high generalization errors.
2. **Overfitting in CNN Models:**  
   Some models are trained on limited data and do not generalize well across different types of deepfakes, making them ineffective in unseen environments.
3. **Lack of Lightweight Deployable Models:**  
   Several proposed solutions are computationally intensive and not optimized for real-time deployment on web or mobile platforms.
4. **Inadequate Visual Explanation or Feedback:**  
   Most systems provide binary classification outputs (Real/Fake) without offering users any confidence scores or visual indicators that explain *why* an image is classified as fake.
5. **Limited Integration with Web Platforms:**  
   Few models are integrated into accessible web interfaces, which limits their usability for the general public or organizations in need of quick authentication.

**TrueSight addresses these gaps by:**

* Building a lightweight CNN model optimized for fast predictions.
* Deploying the system on a user-friendly Flask-based web interface.
* Providing a confidence score along with the classification label.
* Ensuring compatibility with real-time image uploads through a browser.

# CHAPTER 3

# METHODOLOGY

# CHAPTER 3

# METHODOLOGY

## Dataset Description

For the development and evaluation of **TrueSight**, we used a publicly available dataset from **Kaggle**, titled **"Real and Fake Face Detection"**. This dataset is specifically designed to support machine learning models in identifying computer-generated human face images (deepfakes) from real photographs.

**1. Source**

The dataset was downloaded from Kaggle and is maintained by researchers to support experiments in deepfake detection. It contains high-resolution images categorized into **two classes**:

* **Real** – Images of actual human faces captured by cameras.
* **Fake** – Images of synthetic faces generated by **Generative Adversarial Networks (GANs)**, specifically StyleGAN.

**2. Structure**

The dataset is divided into the following folders:

* /real/ – Contains real human face images.
* /fake/ – Contains AI-generated fake human face images.

Each folder contains **JPG** images named numerically or randomly.

**3. Number of Images**

* **Real images**: 95,201
* **Fake images**: 95,134
* **Total**: 190,335 images

The dataset is balanced, which ensures fair training and evaluation without bias toward any class.

**4. Image Resolution**

Each image is in **JPG format**, with a resolution of **64x64** or **128x128 pixels**, making it suitable for input into **Convolutional Neural Networks (CNNs)** without the need for heavy resizing or preprocessing.

**5. Data Preprocessing**

Before training the model, the following preprocessing steps were applied:

* Image normalization (scaling pixel values to 0–1 range)
* Resizing to a uniform input shape (128x128)
* Label encoding (0 for fake, 1 for real)
* Shuffling and splitting into **training (75%)**, **validation (20%)**, and **testing (5%)** sets

**6. Purpose**

This dataset serves as the foundation for training and testing TrueSight's deep learning model. By providing both real and fake face examples, it enables the system to learn the visual features and inconsistencies that distinguish synthetic images from natural ones.

## Data Processing Techniques

To ensure the accuracy and efficiency of the deep learning model in detecting fake and real human face images, proper data processing was carried out before training. The following steps were used to clean, transform, and prepare the dataset downloaded from Kaggle:

**1. Data Loading**

All images were loaded from the **‘real’** and **‘fake’** folders. Labels were assigned to each class:

* 0 for fake images
* 1 for real images

Libraries such as **OpenCV**, **TensorFlow/Keras**, and **NumPy** were used to read and manipulate the images.

**2. Image Resizing**

The original images varied slightly in size. To ensure consistency for input into the Convolutional Neural Network (CNN), all images were resized to a fixed dimension of **128×128 pixels**. This helped reduce computational cost and maintain uniformity.

**Code:**

cv2.resize(image, (128, 128))

**3. Normalization**

Pixel values in images range from 0 to 255. To make training more stable and faster, the pixel values were scaled to the range **[0, 1]** by dividing by 255:

**Code:**

image = image / 255.0

This step is essential for neural networks, especially when using activation functions like ReLU or sigmoid.

**4. Label Encoding**

Labels were converted into a binary format:

* **0** → Fake
* **1** → Real

This helped the model treat the problem as a **binary classification task**.

**5. Dataset Splitting**

The entire dataset was divided into:

* **Training set** – 75% of the data
* **Validation set** – 20% of the data
* **Test set** – 5% of the data

This helped in proper training, tuning, and unbiased performance evaluation.

Python Code:

from sklearn.model\_selection import train\_test\_split

**6. Data Augmentation (Optional)**

To improve generalization and prevent overfitting, optional **data augmentation** techniques were used on the training set:

* Horizontal flipping
* Random rotation
* Zooming
* Shifting

Implemented using **ImageDataGenerator** from Keras:

**Code:**

from tensorflow.keras.preprocessing.image import ImageDataGenerator

This made the model more robust to different types of face orientations and variations.

**7. Shuffling**

Before feeding data into the model, it was shuffled to avoid bias during training. This ensures the model doesn’t learn any order-based patterns.

**8. Conversion to NumPy Arrays or Tensors**

The image data and labels were converted into **NumPy arrays** or **Tensors**, which are the required formats for TensorFlow/Keras models.

**Conclusion**

These data processing techniques helped prepare the dataset effectively for training the deep learning model. Proper preprocessing ensures that the model focuses on learning important visual patterns rather than being affected by noise, resolution mismatch, or data imbalance.

## Model Architecture

The heart of the **TrueSight** system is a **Convolutional Neural Network (CNN)**, a deep learning model specifically designed for visual data such as images. CNNs are highly effective in detecting and classifying patterns by mimicking how the human brain processes visual input.

Our model was implemented using **Python** and the **TensorFlow/Keras** library, and it follows a layered architecture that allows it to learn both low-level and high-level features from images.

**1. Input Layer**

* **Input size:** The model takes an image input of shape **64 × 64 × 3** (width, height, RGB).
* The During preprocessing, images are resized and **normalized (pixel values scaled from 0–255 to 0–1)**.
* **Input Shape in Code:**

input\_shape=(64, 64, 3)

**2. Convolutional + Batch Normalization + MaxPooling Layers**

* These layers are used to extract features, normalize them for stable learning, and reduce spatial dimensions.

**Layer 1**

* **Conv2D:** 32 filters, kernel size 3×3, activation ReLU
* **BatchNormalization:** Normalizes activations for faster convergence
* **MaxPooling2D:** Pool size 2×2

**Code:**

Conv2D(32, (3, 3), activation='relu')

BatchNormalization()

MaxPooling2D((2, 2))

**Layer 2**

* **Conv2D:** 64 filters
* **BatchNormalization**
* **MaxPooling2D**

**Code:**

Conv2D(64, (3, 3), activation='relu')

BatchNormalization()

MaxPooling2D((2, 2))

**Layer 3**

* **Conv2D:** 128 filters
* **BatchNormalization**
* **MaxPooling2D**

**Code:**

Conv2D(128, (3, 3), activation='relu')

BatchNormalization()

MaxPooling2D((2, 2))

**3. Global Average Pooling Layer Layers**

* **Purpose**: Converts each feature map into a single value (average), **reducing parameters and overfitting**.

**Code:**

* GlobalAveragePooling2D()

**4. Fully Connected (Dense) Layer**

* Learns high-level combinations of features.
* 128 neurons with ReLU activation.

**Code:**

Dense(128, activation='relu')

**5. Dropout Layer**

* 50% of the neurons are randomly deactivated during training to **prevent overfitting**.

**Code:**

Dropout(0.5)

**6. Output Layer**

* **dense(1,activation='sigmoid')**  
  Used for binary classification: output is a **probability between 0 and 1**
* Closer to 0 → Fake
* Closer to 1 → Real

**7. Callbacks Used**

* **EarlyStopping**: Stops training early if validation accuracy doesn’t improve for 3 epochs.

**Code:**

EarlyStopping(monitor='val\_acc', patience=3, mode='max', min\_delta=0.01, restore\_best\_weights=True)

* **ReduceLROnPlateau**: Reduces learning rate if validation loss plateaus.

**Code:**

ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=2, min\_lr=1e-6)

**Model Summary Table**

**Layer Type** **Description**  **Output Shape**

Input Layer Input image (64×64×3) (64, 64, 3)

Conv2D 32 filters, 3×3 kernel, ReLU activation (62, 62, 32)

BatchNormalization Normalizes activations (62, 62, 32)

MaxPooling2D Pool size (2×2) (31, 31, 32)

Conv2D 64 filters, 3×3 kernel, ReLU activation (29, 29, 64)

BatchNormalization Normalizes activations (29, 29, 64)

MaxPooling2D Pool size (2×2) (14, 14, 64)

Conv2D 128 filters, 3×3 kernel, ReLU activation (12, 12, 128)

BatchNormalization Normalizes activations (12, 12, 128)

MaxPooling2D Pool size (2×2) (6, 6, 128)

GlobalAvgPooling2D Averages each feature map to a single value(128,)

Dense 128 neurons, ReLU activation (128,)

Dropout 50% neurons dropped (128,)

Output Dense 1 neuron, Sigmoid activation (Binary class) (1,)

**8. Model Compilation**

Once the architecture is defined, we compile the model with the following settings:

Python Code:

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

* **Optimizer**: Adam – an adaptive optimizer that combines the best properties of SGD and RMSProp.
* **Loss Function**: binary\_crossentropy – suitable for binary classification tasks.
* **Metric**: accuracy – used to evaluate the model performance.

**Conclusion**

This CNN model is carefully designed to balance **accuracy**, **efficiency**, and **simplicity**. It is capable of learning complex differences between real and fake faces by extracting hierarchical features from the image. The use of dropout and pooling layers improves generalization, and the binary output provides a clear, interpretable result for classification.

1. **Summary of CNN Layers Workflow**

1. Input: Preprocessed image of size 64×64×3

2. Conv + ReLU + BatchNorm: Extract low-level features

3. Max Pooling: Reduce spatial dimensions

4. Conv + ReLU + BatchNorm → Pooling: Learn deeper features

5. Conv + ReLU + BatchNorm → Pooling: Learn high-level features

6. Global Average Pooling: Compress feature maps into 1D

7. Dense + Dropout: Interpret features and prevent overfitting

8. Output: Final binary prediction using sigmoid (real or fake)

1. **TPU Compatibility**

To improve performance, especially during training, you can run your **TrueSight** image recognition model on a **TPU (Tensor Processing Unit)**. TPUs are hardware accelerators developed by Google specifically for machine learning tasks, and they can **significantly reduce training time** for deep learning models like CNNs.

Here’s how your model is **TPU-compatible**, and what you need to know:

**Why is TrueSight Compatible with TPUs?**

Your model is built using:

* **TensorFlow + Keras**
* Standard layers like Conv2D, MaxPooling2D, Dense, Dropout, Flatten
* No custom or unsupported operations

**All these components are fully supported on TPUs.**

**🚀 Benefits of Running on TPU**

* Up to **10x faster** training than CPU or standard GPU
* Ideal for large datasets (like image datasets from Kaggle)
* Can handle **parallel batch processing** efficiently
* Reduces time-to-train for deep CNN architectures

**⚙️ How to Use TPU on Google Collab**

If you’re using Google Collab (which offers free TPU usage), follow these steps:

1. **Enable TPU**

**CODE:**

# TPU Setup

try:

    resolver = tf.distribute.cluster\_resolver.TPUClusterResolver()  # Detect TPU

    tf.config.experimental\_connect\_to\_cluster(resolver)

    tf.tpu.experimental.initialize\_tpu\_system(resolver)

    strategy = tf.distribute.TPUStrategy(resolver)

    print("✅ TPU initialized.")

except ValueError:

    strategy = tf.distribute.get\_strategy()  # Default strategy (CPU or GPU)

    print("⚠️ TPU not found. Using default strategy.")

print("Number of accelerators:", strategy.num\_replicas\_in\_sync)

**Things to Keep in Mind**

* Use **larger batch sizes** (32, 64, or 128+) to fully utilize TPU capacity.
* Input data pipelines (tf.data.Dataset) should be optimized for performance.
* Avoid using unsupported operations like some custom layers, loops, or NumPy functions in training logic.

**Summary**

| **Feature** | **TPU Compatibility** |
| --- | --- |
| TensorFlow/Keras | ✅ Yes |
| CNN layers (Conv, Pooling) | ✅ Yes |
| Model structure | ✅ TPU-friendly |
| Runtime in Google Collab | ✅ Supported |
| Speed improvement | 🚀 Up to 10x |

1. **Training Strategy (Callback, Early, Stopping, etc.)**

To improve model performance, prevent overfitting, and reduce unnecessary computation, we implemented an efficient training strategy using built-in features from **TensorFlow/Keras**. These strategies help monitor the training process, manage learning behaviour, and automatically save the best version of the model.

**1. Early Stopping**

* **Purpose**: To halt training early when validation accuracy stops improving, avoiding overfitting and saving time.
* **Monitor**: val\_acc (Validation Accuracy)
* **Patience**: 3 epochs
* **Mode**: max (since higher accuracy is better)
* **min\_delta**: 0.01 (minimum change to qualify as improvement)
* **Effect**: Restores the weights from the epoch with the best validation accuracy.

**Code**:

EarlyStopping(

monitor='val\_acc',

patience=3,

mode='max',

min\_delta=0.01,

restore\_best\_weights=True

)

**2. Model Checkpoint**

* **Note**: This callback was not included in the current code. If desired, it can be added to save the best model based on validation performance.

**3. Reduce Learning Rate on Plateau**

* **Purpose**: To reduce the learning rate when training stalls, helping the model converge more effectively.
* **Monitor**: val\_loss
* **Factor**: 0.5 (reduces learning rate by 50%)
* **Patience**: 2 epochs of no improvement
* **Minimum Learning Rate**: 1e-6 (to prevent it from getting too small)

**Code**:

ReduceLROnPlateau(

monitor='val\_loss',

factor=0.5,

patience=2,

min\_lr=1e-6

)

**4. Callback Integration**

All callbacks were added to a list and passed to the model.fit() function to manage training dynamically:

callbacks = [callback, lr\_callback]

**5. Training Configuration Summary**

| **Strategy** | **Purpose** | **Trigger Condition** |
| --- | --- | --- |
| Early Stopping | Stops training when no progress is seen | No improvement in val\_acc |
| Reduce LR on Plateau | Lowers LR to refine learning | Plateau in val\_loss |

**Conclusion**

These training strategies ensured that the model trained efficiently, generalized well to unseen data, and avoided unnecessary computation. By integrating **early stopping** and **adaptive learning rate reduction**, we optimized model convergence while minimizing overfitting — contributing to the robustness of the deepfake detection system.

# CHAPTER 4

# IMPLEMENTATION

**CHAPTER 4**

**IMPLEMENTATION**

## Development Environment

The **TrueSight** project was developed using a cloud-based and open-source environment that supports deep learning and image classification. The tools and platforms used helped in building, training, testing, and evaluating the model efficiently.

**1. Programming Language**

**Python 3.10**  
**Python** is a high-level, general-purpose programming language known for its simplicity, readability, and wide range of applications. It supports multiple programming paradigms, including object-oriented, procedural, and functional programming. Python is widely used in fields such as web development, data analysis, artificial intelligence, and machine learning due to its clean syntax and extensive collection of libraries. In the **TrueSight** project, Python served as the core language for implementing data preprocessing, building the convolutional neural network using TensorFlow, and visualizing results using tools like Matplotlib and Seaborn. Its flexibility and strong community support make Python an ideal choice for developing deep learning applications.

**2. Platform**

**Google Collab** is a free, cloud-based platform provided by Google that allows users to write and run Python code in a browser using a Jupyter Notebook interface. It supports machine learning and data science tasks by offering free access to powerful hardware like GPUs and TPUs, making it ideal for training deep learning models. Collab requires no installation, integrates easily with Google Drive, and enables real-time collaboration, making it a convenient and efficient tool for projects like **TrueSight** that involve training and evaluating neural networks on image data.

**4. Dataset Source**

**Kaggle** is an online platform owned by Google that serves as a hub for data science, machine learning, and artificial intelligence communities. It offers a wide range of public datasets, coding competitions, and collaborative tools that allow users to explore, analyse, and share data-driven projects. Kaggle provides free access to notebooks (similar to Jupyter), GPU resources, and an active community of professionals and learners. For the **TrueSight** project, Kaggle was used as the source for the dataset containing real and fake human face images. The availability of clean, labelled datasets made it easier to train and evaluate the model effectively. Kaggle continues to be a valuable resource for both beginners and experts in the AI field.

**5. Hardware**

* **TPU or GPU** via Google Collab was used to speed up training.
* No local setup or installation was needed.

**6. File Management**

* **Google Drive** was used to store datasets, notebooks, model files, and training outputs for backup and sharing.

## LIBRARIES AND FRAMEWORKS

To implement the TrueSight image recognition system for detecting fake human faces, several powerful Python libraries and frameworks were used. These tools helped in building the deep learning model, processing image data, visualizing performance, and evaluating results.

**1. TensorFlow**

* **Purpose**: Deep Learning Framework
* **Description**: TensorFlow is an open-source machine learning framework developed by Google. It allows users to build and train deep neural networks with flexibility and efficiency. In the TrueSight project, TensorFlow was used to define, train, and evaluate the Convolutional Neural Network (CNN) for classifying images as real or fake.
* **Key Features Used**: Model building (Sequential API), optimizers, loss functions, and callbacks.

**2. Keras (via TensorFlow)**

* **Purpose**: High-Level Deep Learning API
* **Description**: Keras is an easy-to-use API built into TensorFlow. It simplifies the process of creating neural networks by providing a user-friendly interface. It was used to stack layers like Conv2D, MaxPooling2D, Flatten, Dense, and Dropout in our CNN model.
* **Key Features Used**: Model definition, training (model.fit), and evaluation functions.

**3. NumPy**

* **Purpose**: Numerical Computation
* **Description**: NumPy is the foundational library for numerical operations in Python. It provides support for large multi-dimensional arrays and matrices. In this project, it was used to convert image data into arrays, normalize pixel values, and manage datasets during preprocessing.
* **Key Functions Used**: np.array, np.reshape, and basic array math.

**4. OpenCV (cv2)**

* **Purpose**: Image Processing
* **Description**: OpenCV is an open-source computer vision library. It was used to read images, resize them to a uniform shape (128×128), and convert them into a suitable format for training. OpenCV helped streamline image preprocessing tasks efficiently.
* **Key Functions Used**: cv2.imread, cv2.resize, cv2.cvtColor.

**5. Matplotlib**

* **Purpose**: Data Visualization
* **Description**: Matplotlib is a widely-used library for creating static, animated, and interactive plots. It was used in TrueSight to visualize training and validation accuracy, loss over epochs, and final prediction results.
* **Key Features Used**: plt.plot, plt.title, plt.xlabel, plt.ylabel.

**6. Seaborn**

* **Purpose**: Advanced Statistical Visualization
* **Description**: Built on top of Matplotlib, Seaborn provides aesthetically pleasing and informative visualizations. In TrueSight, it was mainly used to draw the **confusion matrix**, which helped analyze the performance of the classification model.
* **Key Functions Used**: sns.heatmap.

**7. Scikit-learn (sklearn)**

* **Purpose**: Machine Learning Utilities
* **Description**: Scikit-learn is a powerful library for machine learning tasks and evaluation metrics. It was used for splitting the dataset into training and test sets and for evaluating the model’s performance using metrics such as accuracy, precision, recall, and F1-score.
* **Key Modules Used**: train\_test\_split, classification\_report, confusion\_matrix.

**8. Google Collab Environment**

* **Purpose**: Cloud-based Coding and Execution
* **Description**: Google Collab is an online platform that allows users to run Python code in a Jupyter Notebook-style environment with free access to GPUs and TPUs. It was used for writing and running the full project code, training the model, storing files on Google Drive, and visualizing results.

## Model Training

**Dataset Preparation**

* **Real images**: Collected from legitimate sources such as CelebA and FFHQ. These datasets contain high-quality, labeled images of celebrities and diverse faces, ideal for training a deepfake detection model.
* **Fake images**: Generated using deepfake generators like StyleGAN and ThisPersonDoesNotExist. These serve as the fake images for the binary classification task (real vs. fake).
* **Class Balance**: The dataset is balanced to ensure that both classes (real and fake) are represented equally, preventing model bias.

**Data Preprocessing**

* **Resize images**: Images are resized to 64x64 pixels to match the model’s input shape of (64, 64, 3), which helps reduce computational load while retaining image quality.
* **Normalization**: Pixel values are normalized to a range between 0 and 1 by dividing by 255.
* **Augmentation**: Random transformations (e.g., rotation, flipping) are applied to the images to increase variability and improve model robustness.

**Model Selection**

* **CNN Architecture**: The model consists of:
  + **Conv2D Layer**: The first layer uses 32 filters of size (3, 3) with ReLU activation.
  + **BatchNormalization**: Applied after each convolution to stabilize training.
  + **MaxPooling2D Layer**: Max pooling (2x2) reduces spatial dimensions and computational complexity.
  + **Additional Conv2D Layers**: Two more convolutional layers with 64 and 128 filters, followed by batch normalization and max pooling layers.
  + **GlobalAveragePooling2D**: This layer reduces the spatial dimensions to a single value per feature map.
  + **Dense Layer**: A fully connected layer with 128 neurons and ReLU activation.
  + **Dropout Layer**: Applied after the dense layer with a dropout rate of 50% to prevent overfitting.
  + **Output Layer**: A single neuron with a sigmoid activation function for binary classification.

**Training**

* **Loss Function**: Binary cross-entropy loss is used for binary classification tasks.
* **Optimizer**: Adam optimizer is used for efficient training.
* **Metrics**: The model tracks accuracy (acc) during training.
* **Callbacks**:
  + **EarlyStopping**: Monitors val\_acc with a patience of 3 epochs. The model stops early if validation accuracy does not improve, and the best weights are restored.
  + **ReduceLROnPlateau**: Reduces the learning rate by a factor of 0.5 if the validation loss plateaus for 2 epochs, helping the model converge more smoothly.

**Training Process**

**Code:**

history = baseline\_model.fit(

X\_train, y\_train, # Training data

batch\_size=128, # Number of samples per gradient update

epochs=25, # Number of epochs

validation\_data=(val\_x, val\_y), # Validation data

callbacks=[callback, lr\_callback] # Callbacks for early stopping and learning rate adjustment

)

This training configuration ensures that the model learns efficiently, prevents overfitting, and maximizes performance through the use of callbacks, appropriate loss functions, and an effective architecture.

**4.4 Model Evaluation**

**Confusion Matrix**

* Helps you visualize:
  + True Positives (real identified as real)
  + False Positives (fake identified as real)
  + etc.

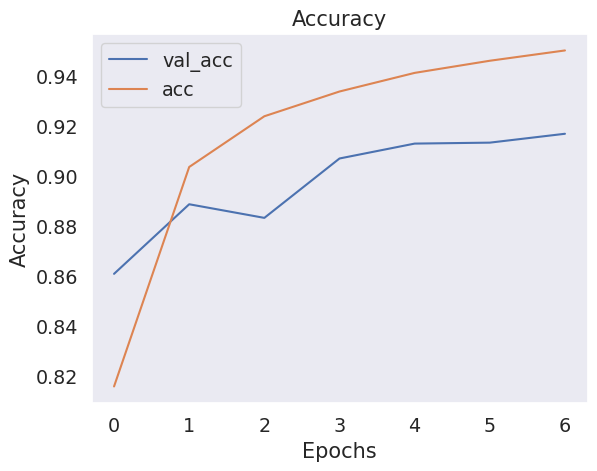
**Metrics**

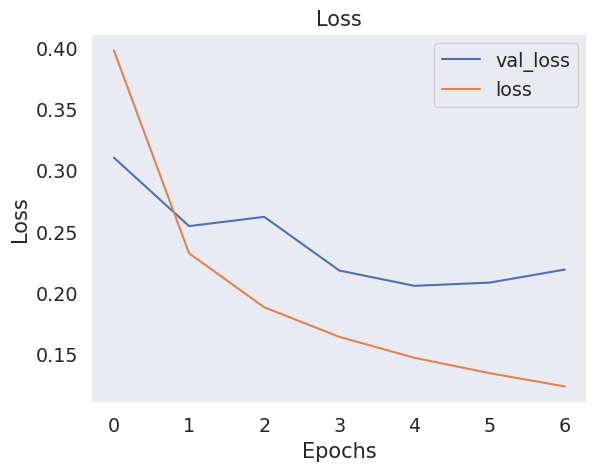
* Accuracy: (TP + TN) / total
* Precision: TP / (TP + FP) (important if false alarms are costly)
* Recall: TP / (TP + FN) (important if missing fakes is risky)
* F1 Score: Harmonic mean of Precision & Recall
* ROC Curve & AUC: Evaluates trade-off across thresholds

**Real-World Testing**

* Test on out-of-distribution images (e.g., new GANs).
* Check robustness to image perturbations or compression.

**4.5VISUALIZATION: ACCURACY & LOSS GRAPHS**

****



# CHAPTER 5

# CODE SNAPSHOTS AND OUTPUT

**CHAPTER 5**

**WEB DEPLOYMENT (TrueSight)**

## 5.1 FLASK FRAMEWORK OVERVIEW

Flask is a lightweight Python web framework used for building web applications quickly and with minimal code. It is ideal for deploying machine learning models due to its flexibility and simplicity. In this project, Flask acts as the backend server that handles HTTP requests, routes them appropriately, and connects the trained model with the frontend interface.

Key features:

* Micro-framework with minimal setup.
* Integrated development server and debugger.
* RESTful request dispatching.
* Support for Jinja2 templating engine.

**Code:App.py**

from flask import Flask, render\_template, request, jsonify, redirect, session

import os

from PIL import Image

import numpy as np

import tensorflow as tf

*# ✅ AI Chat Setup (preserved as requested)*

from transformers import pipeline

chat\_model = pipeline("text-generation", model="facebook/blenderbot-400M-distill")

try:

    import g4f

    use\_g4f = True

except ImportError:

    use\_g4f = False

*# Initialize Flask app*

app = Flask(\_\_name\_\_)

UPLOAD\_FOLDER = 'static/uploads'

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

app.secret\_key = os.urandom(24)

*# ⏳ Load Trained CNN Model*

print("⏳ Loading CNN model and weights...")

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input\_shape=(128, 128, 3)),

    MaxPooling2D((2, 2)),

    Conv2D(16, (3, 3), activation='relu'),

    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(1, activation='sigmoid')

])

model.compile(optimizer='RMSprop', loss='binary\_crossentropy', metrics=['accuracy'])

*# ✅ Load model weights (update path if needed)*

model.load\_weights('C:/Users/acer/Downloads/new\_model.h5', by\_name=True, skip\_mismatch=True)

print("✅ Model loaded and ready.")

*# 🧠 Prediction Function*

def predict\_deepfake(image\_path):

    image = Image.open(image\_path).convert("RGB")

    image = image.resize((128, 128))

    image\_array = np.array(image) / 255.0

    image\_array = np.expand\_dims(image\_array, axis=0)

    prediction = model.predict(image\_array)[0][0]

    label = "Authenticated" if prediction >= 0.5 else "Manipulated"

    return label, float(prediction)

*# 🔗 Flask Routes*

@app.route('/')

def home():

    return render\_template('home.html')

@app.route('/about')

def about():

    return render\_template('about.html')

@app.route('/blogs')

def blogs():

    return render\_template('blogs.html')

@app.route('/upload', methods=['GET', 'POST'])

def upload():

    if request.method == 'POST':

        uploaded\_file = request.files['file']

        if uploaded\_file:

            file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], uploaded\_file.filename)

            uploaded\_file.save(file\_path)

            label, confidence = predict\_deepfake(file\_path)

            session['image\_path'] = uploaded\_file.filename

            session['label'] = label

            session['confidence'] = round(confidence \* 100, 2)

            return redirect('/result')

    return render\_template('upload.html')

@app.route('/result')

def result():

    image\_path = session.get('image\_path', '')

    label = session.get('label', 'Unknown')

    confidence = session.get('confidence', 0)

    return render\_template('result.html', image\_path=image\_path, label=label, confidence=confidence)

*# 🤖 AI Chat Endpoint*

@app.route('/chat', methods=['POST'])

def chat():

    data = request.get\_json()

    user\_message = data.get("message")

    if not user\_message:

        return jsonify({"response": "Please enter a message."})

    try:

        print(f"📩 User Input: {user\_message}")

        if use\_g4f:

            response = g4f.ChatCompletion.create(

                model=g4f.models.default,

                messages=[{"role": "user", "content": user\_message}]

            )

        else:

            response = chat\_model(user\_message, max\_length=100)[0]["generated\_text"]

        print(f"🤖 AI Response: {response}")

        return jsonify({"response": response})

    except Exception as e:

        print(f"⚠️ Error: {str(e)}")

        return jsonify({"response": "An error occurred while generating a response."})

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**5.2 FRONTEND DESIGN (HTML, CSS)**

The frontend of the TrueSight application is built using HTML and CSS, offering a simple and user-friendly interface. It allows users to:

* Upload an image for deepfake detection.
* View the prediction result (Real or Fake).
* See output accuracy and relevant messages.

Design Elements:

* HTML forms for image upload.
* CSS styling for layout, fonts, and responsiveness.
* A results section to display prediction outcomes.

**Code:**

**Base.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>DeepFake Detect</title>

*<!-- External CSS -->*

    <link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

*<!-- Bootstrap CSS -->*

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

*<!-- Font Awesome for Chat Icon -->*

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css">

</head>

<body>

*<!-- Navbar -->*

    <nav class="navbar">

        <div class="container-fluid">

            <a class="navbar-brand" href="/">

                <img src="{{ url\_for('static', filename='images/logo.png') }}" alt="Logo">

                <span>TrueSight</span>

            </a>

            <ul class="navbar-nav ms-auto d-flex flex-row">

                <li class="nav-item"><a class="nav-link" href="/">Home</a></li>

                <li class="nav-item"><a class="nav-link" href="/blocks">Blogs</a></li>

                <li class="nav-item"><a class="nav-link" href="/about">About Us</a></li>

            </ul>

        </div>

    </nav>

*<!-- Chatbox (Hidden by default) -->*

    <div id="chatbox" class="chat-popup">

        <div class="chat-header">

            <h5>AI Chatbot</h5>

            <button onclick="closeChat()" class="close-btn">&times;</button>

        </div>

        <div class="chat-body" id="chat-body">

            <p class="bot-message">Hello! How can I help you?</p>

        </div>

        <div class="chat-footer">

            <input type="text" id="user-input" placeholder="Type a message..." onkeypress="sendMessage(event)">

            <button onclick="sendMessage(event)">Send</button>

        </div>

    </div>

*<!-- Chat Icon to Open Chat -->*

    <button id="chat-toggle" onclick="openChat()">

        <i class="fa-solid fa-robot" style="color: #0d067a;"></i>

    </button>

*<!-- Main Content -->*

    <div class="container mt-4">

        {% block content %}{% endblock %}

    </div>

*<!-- Footer -->*

    <footer class="footer mt-auto py-3 bg-light">

        <div class="container text-center">

            <p>© 2024 TrueSight. All Rights Reserved.</p>

            <p>

                <a href="/privacy-policy" class="footer-link">Privacy Policy</a> |

                <a href="/terms" class="footer-link">Terms of Service</a> |

                <a href="/contact" class="footer-link">Contact Us</a>

            </p>

        </div>

    </footer>

*<!-- JavaScript for Chatbox -->*

    <script>

        function openChat() {

            document.getElementById("chatbox").style.display = "block";

            document.getElementById("chat-toggle").style.display = "none"; *// Hide icon when chat is open*

        }

        function closeChat() {

            document.getElementById("chatbox").style.display = "none";

            document.getElementById("chat-toggle").style.display = "block"; *// Show icon when chat is closed*

        }

        async function sendMessage(event) {

            if (event.key === "Enter" || event.type === "click") {

                let userInput = document.getElementById("user-input").value.trim();

                if (userInput === "") return;

                let chatBody = document.getElementById("chat-body");

*// Display User Message*

                let userMessage = document.createElement("p");

                userMessage.className = "user-message";

                userMessage.innerText = userInput;

                chatBody.appendChild(userMessage);

                document.getElementById("user-input").value = "";

                try {

                    let response = await fetch("/chat", {

                        method: "POST",

                        headers: { "Content-Type": "application/json" },

                        body: JSON.stringify({ message: userInput })

                    });

                    let data = await response.json();

                    let botMessage = document.createElement("p");

                    botMessage.className = "bot-message";

                    botMessage.innerText = data.response;

                    chatBody.appendChild(botMessage);

                } catch (error) {

                    console.error("Error:", error);

                    let errorMessage = document.createElement("p");

                    errorMessage.className = "bot-message";

                    errorMessage.innerText = "Oops! Something went wrong.";

                    chatBody.appendChild(errorMessage);

                }

                chatBody.scrollTop = chatBody.scrollHeight;

            }

        }

    </script>

*<!-- Chatbox & Icon CSS -->*

    <style>

        /\* Chatbox Styling \*/

        .chat-popup {

            position: fixed;

            bottom: 20px;

            right: 20px;

            width: 300px;

            background: white;

            border: 1px solid #ccc;

            box-shadow: 0px 0px 10px rgba(13, 22, 155, 0.2);

            border-radius: 10px;

            overflow: hidden;

            z-index: 1000;

            display: none; /\* Start hidden \*/

        }

        .chat-header {

            background: #0d067a;

            color: white;

            padding: 10px;

            display: flex;

            justify-content: space-between;

        }

        .chat-body {

            padding: 10px;

            max-height: 250px;

            overflow-y: auto;

        }

        .chat-footer {

            display: flex;

            border-top: 1px solid #ccc;

            padding: 5px;

        }

        .chat-footer input {

            flex: 1;

            border: none;

            padding: 8px;

            outline: none;

            cursor: text;

        }

        .chat-footer button {

            border: none;

            padding: 8px;

            background: #0d067a;

            color: white;

            cursor: pointer;

            transition: 0.3s;

        }

        .chat-footer button:hover {

            background:rgb(206, 205, 221);

            color: #0d067a;

        }

        .user-message, .bot-message {

            padding: 5px;

            border-radius: 5px;

            margin: 5px 0;

        }

        .user-message {

            background: #dcf8c6;

            text-align: right;

        }

        .bot-message {

            background: #f1f1f1;

            text-align: left;

        }

        .close-btn {

            background: transparent;

            border: none;

            color: white;

            font-size: 18px;

            cursor: pointer;

        }

        #chat-toggle {

            position: fixed;

            bottom: 20px;

            right: 20px;

            background: rgb(206, 205, 221);

            color: white;

            border: none;

            padding: 15px;

            border-radius: 50%;

            cursor: pointer;

            font-size: 20px;

            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

            transition: 0.3s;

            display: flex; /\* Ensures icon stays centered \*/

            align-items: center;

            justify-content: center;

        }

        #chat-toggle i {

            font-size: 28px; /\* Increase icon size \*/

        }

        #chat-toggle:hover {

            background: rgb(206, 205, 221);

        }

    </style>

</body>

</html>

**Home.html**

{% extends "base.html" %} {% block content %}

<br><br><br>

<div class="container mt-4">

  <div class="row align-items-center">

*<!-- Left side: Heading, description, and button -->*

    <div class="col-md-6 text-center text-md-start">

      <h1 class="display-4">TrueSight: Image Authentication</h1>

      <p>Upload an image to test for possible deepfakes</p>

      <a

      href="{{ url\_for('upload') }}"

      class="btn btn-primary btn-lg"

      style="

        font-size: 1rem;

        padding: 10px 20px;

        border: none; /\* No border \*/

        border-radius: 25px;

        background-color: #28a745;

        color: #fff; /\* Green text \*/

        box-shadow: 0 4px 10px rgba(0, 0, 0, 0.15);

        transition: transform 0.2s, box-shadow 0.2s, background-color 0.3s ease, color 0.3s ease;

      "

      onmouseover="this.style.boxShadow='0 6px 15px rgba(0, 0, 0, 0.25)'; this.style.transform='translateY(-3px)'; this.style.backgroundColor='#fff'; this.style.color='#28a745';"

      onmouseout="this.style.boxShadow='0 4px 10px rgba(0, 0, 0, 0.15)'; this.style.transform='translateY(0)'; this.style.backgroundColor='#28a745'; this.style.color='#fff';"

        >Get Started</a

      >

    </div>

*<!-- Right side: GIF -->*

    <div class="col-md-6 text-center">

      <img

        src="{{ url\_for('static', filename='images/deepfake\_detect.gif') }}"

        alt="Deepfake Detection GIF"

        class="img-fluid rounded shadow"

        style="max-width: 100%; height: auto;"

      />

    </div>

  </div>

  <h2 class="text-center my-4">Samples</h2>

  <div id="samplesCarousel" class="carousel slide" data-bs-ride="carousel">

    <div class="carousel-inner">

      <div class="carousel-item active">

        <div class="d-flex justify-content-center">

          <img

            src="{{ url\_for('static', filename='images/sample1.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

            border-radius="4"

          />

          <img

            src="{{ url\_for('static', filename='images/sample2.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample3.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample4.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample5.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample6.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample7.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample1.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample2.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample3.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample4.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample5.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample6.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample7.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

        </div>

      </div>

      <div class="carousel-item">

        <div class="d-flex justify-content-center">

          <img

            src="{{ url\_for('static', filename='images/sample1.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample2.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample3.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample4.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample5.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample6.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample7.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample1.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample2.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          <img

            src="{{ url\_for('static', filename='images/sample3.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample4.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample5.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample6.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

          <img

            src="{{ url\_for('static', filename='images/sample7.png') }}"

            class="img-thumbnail mx-2"

            width="222"

            height="300"

          />

        </div>

      </div>

    </div>

*<!-- Carousel controls -->*

    <button

      class="carousel-control-prev"

      type="button"

      data-bs-target="#samplesCarousel"

      data-bs-slide="prev"

    >

      <span class="carousel-control-prev-icon" aria-hidden="true"></span>

      <span class="visually-hidden">Previous</span>

    </button>

    <button

      class="carousel-control-next"

      type="button"

      data-bs-target="#samplesCarousel"

      data-bs-slide="next"

    >

      <span class="carousel-control-next-icon" aria-hidden="true"></span>

      <span class="visually-hidden">Next</span>

    </button>

  </div>

</div>

{% endblock %}

Upload.html

{% extends "base.html" %}

{% block content %}

<div class="custom-upload-page container mt-5">

    <style>

        /\* Centered upload box \*/

        .upload-section {

    background: white;

    padding: 30px;

    border-radius: 15px;

    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);

    max-width: 600px;

    width: 100%;

    text-align: center;

    position: absolute; /\* Absolute positioning \*/

    top: 50%;

    left: 50%;

    transform: translate(-50%, -50%); /\* Centering Trick \*/

}

        .custom-upload-page h2 {

            color: #333;

            font-size: 2rem;

            font-weight: bold;

            margin-bottom: 20px;

        }

        .form-group {

            position: relative;

            display: flex;

            align-items: center;

        }

        /\* File input \*/

        .form-group input[type="file"] {

            width: 100%;

            padding: 12px;

            border: 2px dashed #6c757d;

            border-radius: 8px;

            background-color: #f9f9f9;

            text-align: center;

        }

        .form-group input[type="file"]:hover {

            border-color: #28a745;

            background-color: #eafbee;

        }

        /\* Camera Icon - Positioned on the right \*/

        .camera-icon {

            position: absolute;

            right: 10px;

            top: 50%;

            transform: translateY(-50%);

            background-color:rgb(0, 89, 255);

            color: white;

            border: none;

            padding: 10px;

            border-radius: 50%;

            cursor: pointer;

            font-size: 18px;

            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

            transition: 0.3s;

        }

        .camera-icon:hover {

            background: #0056b3;

            transform: translateY(-50%) scale(1.1);

        }

        /\* Camera Modal (Dialog Box) \*/

        .modal {

            display: none;

            position: fixed;

            z-index: 1000;

            left: 0;

            top: 0;

            width: 100%;

            height: 100%;

            background-color: rgba(0, 0, 0, 0.5);

            justify-content: center;

            align-items: center;

        }

        .modal-content {

            background:rgb(230, 229, 243);

            padding: 20px;

            border-radius: 10px;

            text-align: center;

            width: 80%;

            max-width: 400px;

            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.3);

            position: relative;

        }

        #video {

            width: 100%;

            border-radius: 10px;

            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.15);

        }

        #capture-btn {

            margin-top: 10px;

            background-color: #28a745;

            color: white;

            padding: 10px 15px;

            border: none;

            border-radius: 5px;

            cursor: pointer;

        }

        #capture-btn:hover {

            background-color: #1e7e34;

        }

        .close-modal {

            background: red;

            color: white;

            padding: 5px 10px;

            border: none;

            border-radius: 5px;

            cursor: pointer;

            position: absolute;

            top: 10px;

            right: 10px;

        }

        /\* Upload Button Styling \*/

.btn {

    font-size: 1rem;

    padding: 12px 25px;

    border-radius: 25px;

    background-color: #28a745; /\* Green color \*/

    color: white;

    border: none;

    transition: all 0.3s ease-in-out;

    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.15);

    cursor: pointer;

}

/\* Hover Effect \*/

.btn:hover {

    transform: translateY(-3px); /\* Slight lift \*/

    box-shadow: 0 6px 15px rgba(0, 0, 0, 0.25);

    background-color: white;

    color: #28a745;

}

    </style>

    <div class="upload-section">

        <h2>Upload an Image</h2>

        <form id="upload-form" action="{{ url\_for('upload') }}" method="POST" enctype="multipart/form-data">

            <div class="form-group">

                <input type="file" name="file" class="form-control-file" id="file-input" required>

*<!-- Camera Icon -->*

                <button type="button" class="camera-icon" onclick="openModal()">

                    <i class="fa-solid fa-camera"></i>

                </button>

            </div>

            <button type="submit" class="btn mt-3">Upload</button>

        </form>

    </div>

</div>

*<!-- Camera Modal -->*

<div id="camera-modal" class="modal">

    <div class="modal-content">

        <button class="close-modal" onclick="closeModal()">&times;</button>

        <h2>Capture Image</h2>

        <video id="video" autoplay></video>

        <canvas id="canvas" style="display: none;"></canvas>

        <button id="capture-btn">Capture</button>

    </div>

</div>

<script>

    const video = document.getElementById('video');

    const canvas = document.getElementById('canvas');

    const captureBtn = document.getElementById('capture-btn');

    const uploadForm = document.getElementById('upload-form');

    const fileInput = document.getElementById('file-input');

    const modal = document.getElementById('camera-modal');

*// Open modal and start camera*

    function openModal() {

        modal.style.display = "flex";

        navigator.mediaDevices.getUserMedia({ video: true })

            .then((stream) => {

                video.srcObject = stream;

            })

            .catch((err) => {

                console.error("Error accessing camera:", err);

            });

    }

*// Close modal and stop camera*

    function closeModal() {

        modal.style.display = "none";

        let tracks = video.srcObject ? video.srcObject.getTracks() : [];

        tracks.forEach(track => track.stop());

        video.srcObject = null;

    }

*// Capture image and set as file input*

    captureBtn.addEventListener('click', function() {

        const context = canvas.getContext('2d');

        canvas.width = video.videoWidth;

        canvas.height = video.videoHeight;

        context.drawImage(video, 0, 0, canvas.width, canvas.height);

        canvas.toBlob((blob) => {

            let file = new File([blob], "captured\_image.jpg", { type: "image/jpeg" });

            let dataTransfer = new DataTransfer();

            dataTransfer.items.add(file);

            fileInput.files = dataTransfer.files;

            alert("Image Captured! Now press Upload.");

            closeModal();

        }, "image/jpeg");

    });

*// Close modal when clicking outside*

    window.onclick = function(event) {

        if (event.target === modal) {

            closeModal();

        }

    };

</script>

{% endblock %}

Result.html

{% extends "base.html" %}

{% block content %}

<div class="container mt-5 result-page">

    <style>

        /\* CSS specific to result.html \*/

        body {

            background: linear-gradient(to right, #fffffe, #c4c5c7);

            font-family: 'Arial', sans-serif;

            color: #333;

        }

        .result-page {

            max-width: 700px;

            margin: auto;

            padding: 0 15px; /\* To ensure responsiveness \*/

        }

        .result-page .text-center {

            background-color: rgba(255, 255, 255, 0.95);

            color: #333;

            padding: 30px;

            border-radius: 15px;

            box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);

            max-width: 600px; /\* Fixed width for the container \*/

            margin: 0 auto; /\* Center horizontally \*/

        }

        .result-page h2 {

            font-size: 2.5rem;

            margin-bottom: 20px;

            color: #2c3e50;

        }

        .result-page .img-thumbnail {

            border: 4px solid #2c3e50;

            border-radius: 15px;

            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.3);

            transition: transform 0.3s ease, box-shadow 0.3s ease;

        }

        .result-page .img-thumbnail:hover {

            transform: scale(1.05);

            box-shadow: 0 6px 20px rgba(0, 0, 0, 0.4);

        }

        .result-page .mt-3 p {

            font-size: 1.2rem;

            font-weight: 500;

            color: #2c3e50;

        }

        .result-page .btn-primary {

            background-color: #28a745;

            border: none;

            font-size: 1.2rem;

            padding: 12px 30px;

            color: #fff;

            border-radius: 25px;

            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);

            transition: transform 0.2s, box-shadow 0.2s;

        }

        .result-page .btn-primary:hover {

            transform: translateY(-3px);

            box-shadow: 0 6px 15px rgba(0, 0, 0, 0.25);

            background-color: #fff;

            color: #28a745;

        }

    </style>

    <div class="text-center">

        <h2>Prediction Results</h2>

        <img src="{{ url\_for('static', filename='uploads/' + image\_path) }}" class="img-thumbnail" width="300">

        <div class="mt-3">

            <p><strong>Result:</strong> {{ label }}</p>

*<!---->*  <p><strong>Confidence:</strong> {{ confidence | round(2) }}%</p>

        </div>

        <a href="/" class="btn btn-primary mt-4">Go Home</a>

    </div>

</div>

{% endblock %}

**5.3 BACKEND LOGIC FOR MODEL INTEGRATION**

The backend logic uses Flask to connect the trained CNN model to the web interface. Key operations:

* Load the saved Keras model on server startup.
* Accept user-uploaded images through the HTML form.
* Preprocess the image (resize to 64×64, normalize).
* Perform prediction using the loaded model.
* Return results (e.g., "Real Image" or "Fake Image") to the user.

Code Snippet:

from keras.models import load\_model

# Load the trained model (use .keras if saved in native format)

model = load\_model('/content/drive/MyDrive/TrueSight/model5.h5')

from PIL import Image

import numpy as np

def predict\_image(image\_path, model):

    # Load and preprocess the image

    image = Image.open(image\_path).convert('RGB')

    image = image.resize((64, 64))  # Resize as per model input

    image\_array = np.array(image) / 255.0  # Normalize

    image\_array = np.expand\_dims(image\_array, axis=0)  # Add batch dimension

    # Predict

    prediction = model.predict(image\_array)[0][0]

    label = "Real" if prediction > 0.5 else "Fake"

    confidence = prediction if prediction > 0.5 else 1 - prediction

    print(f" Prediction: {label} | Confidence: {confidence \* 100:.2f}%")

from google.colab import files

uploaded = files.upload()  # Upload image

for image\_path in uploaded.keys():

    predict\_image(image\_path, model)

@app.route('/result')

def result():

    image\_path = session.get('image\_path', '')

    label = session.get('label', 'Unknown')

    confidence = session.get('confidence', 0) \* 100

    return render\_template('result.html', image\_path=image\_path, label=label, confidence=confidence)

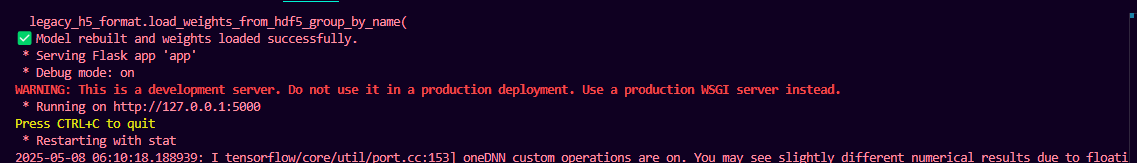
**5.4 IMAGE UPLOAD & PREDICTION FLOW**

1. User Action: The user uploads an image using the website interface.
2. Image Handling: Flask receives the image via POST request.
3. Preprocessing: The image is resized to (64, 64), converted to an array, normalized, and reshaped.
4. Prediction: The model predicts the class (0 = Real, 1 = Fake).
5. Response: The result is displayed back on the web page with a relevant message.

**5.5 LOCAL WEBSITE TESTING (TrueSight)**

The application was tested locally using Flask’s development server:

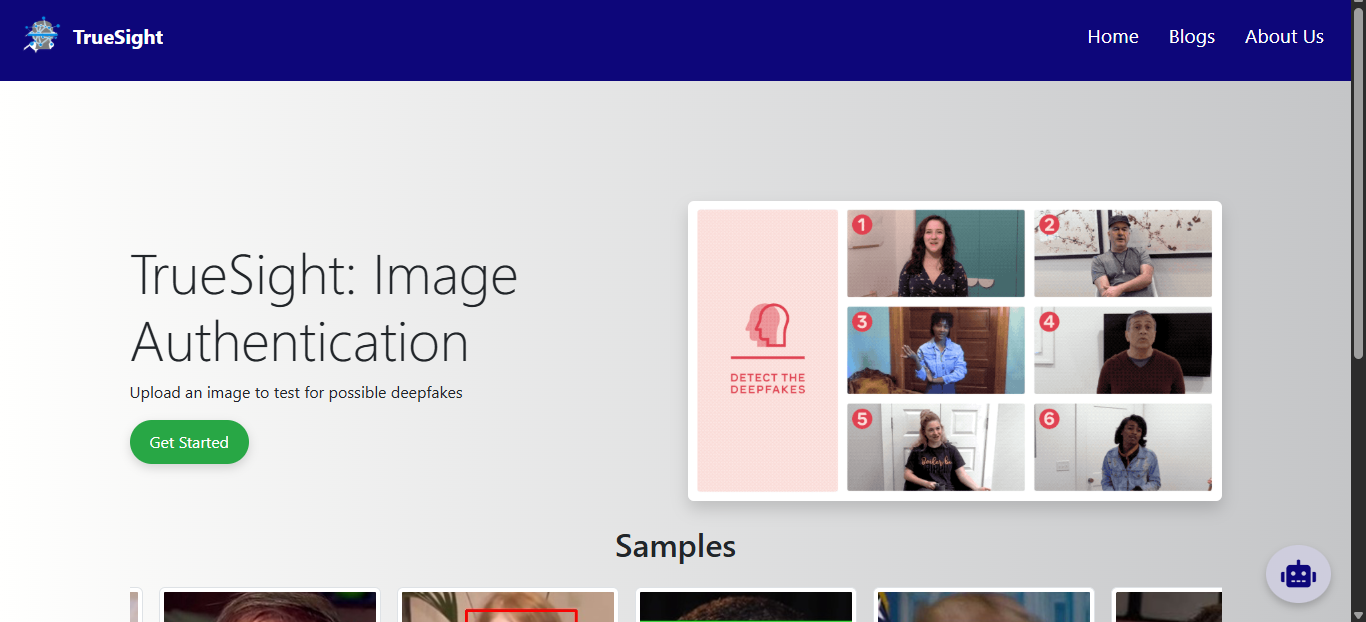
* Run command: python app.py
* Accessed via: http://localhost:5000
* Validated functionalities: image upload, real-time prediction, and response generation.
* Tested on multiple browsers for responsiveness and compatibility.



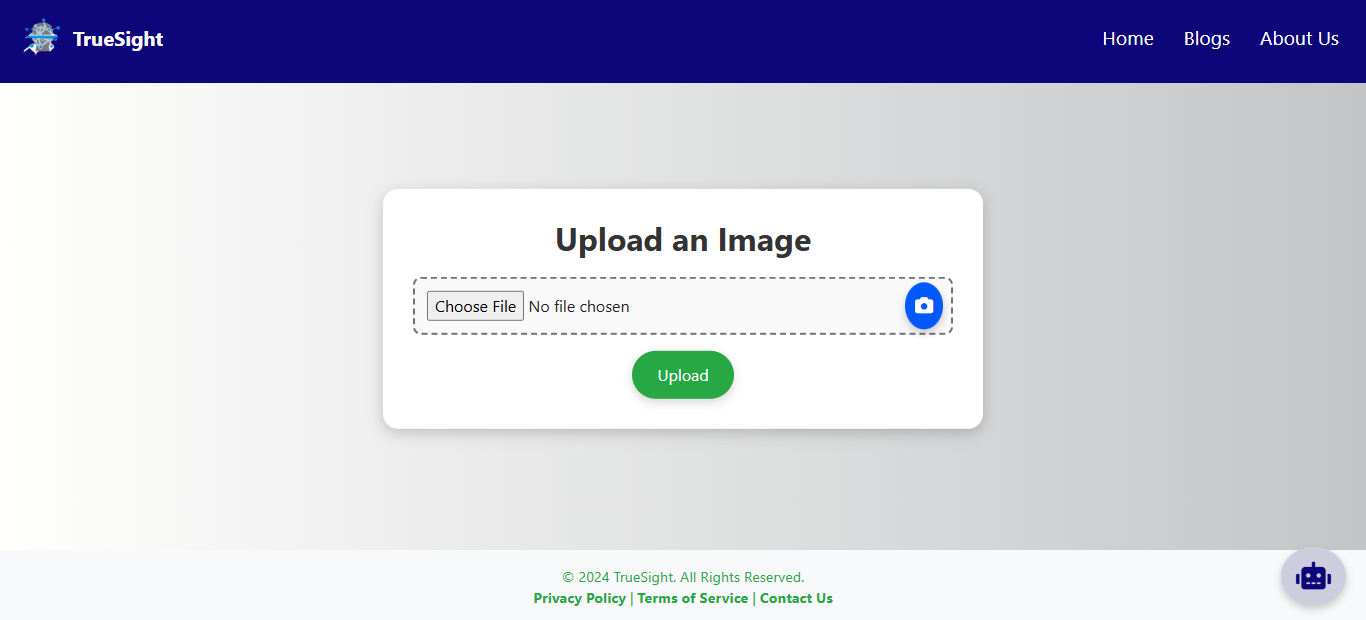
**Figure 5.1.1) Register Page**

**5.6 OUTPUT SCREENSHOTS**

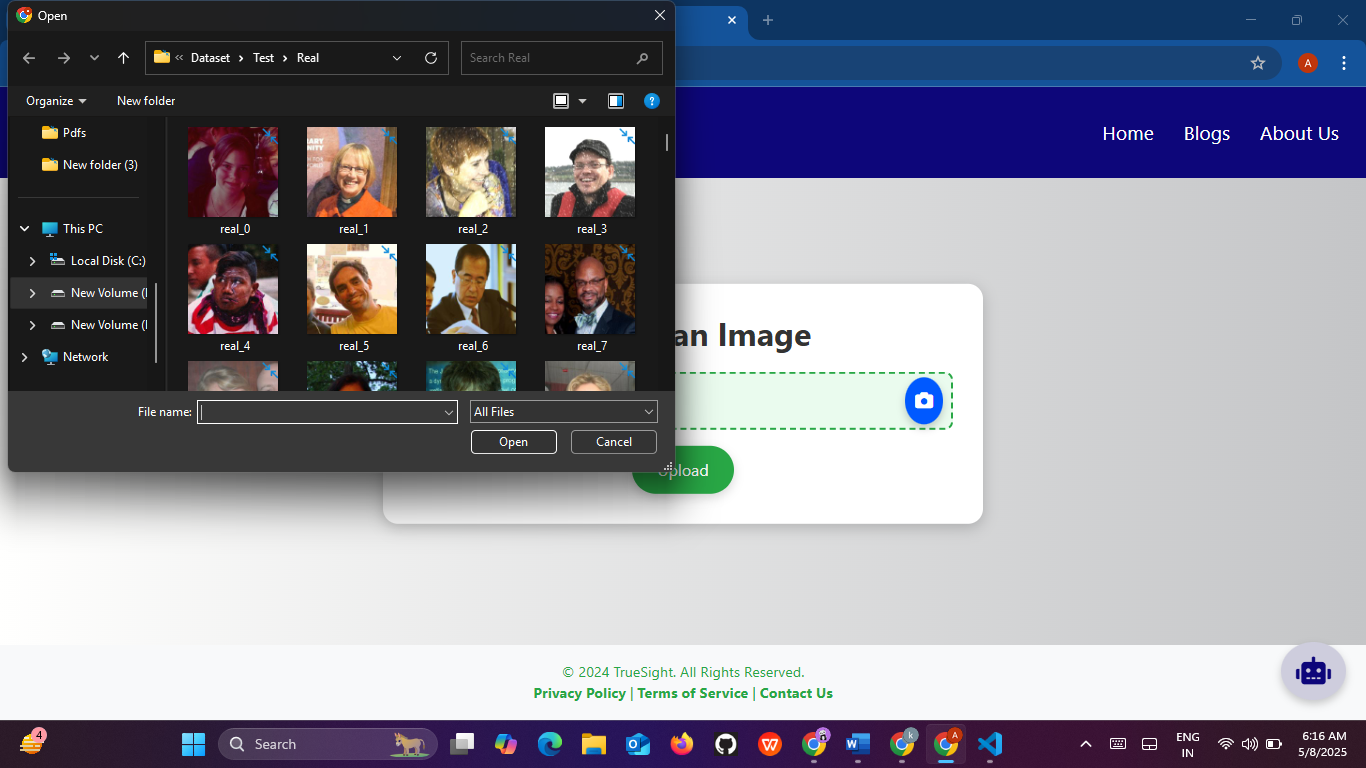
**5.6.1) TrueSight Image Authentication Output:**

****

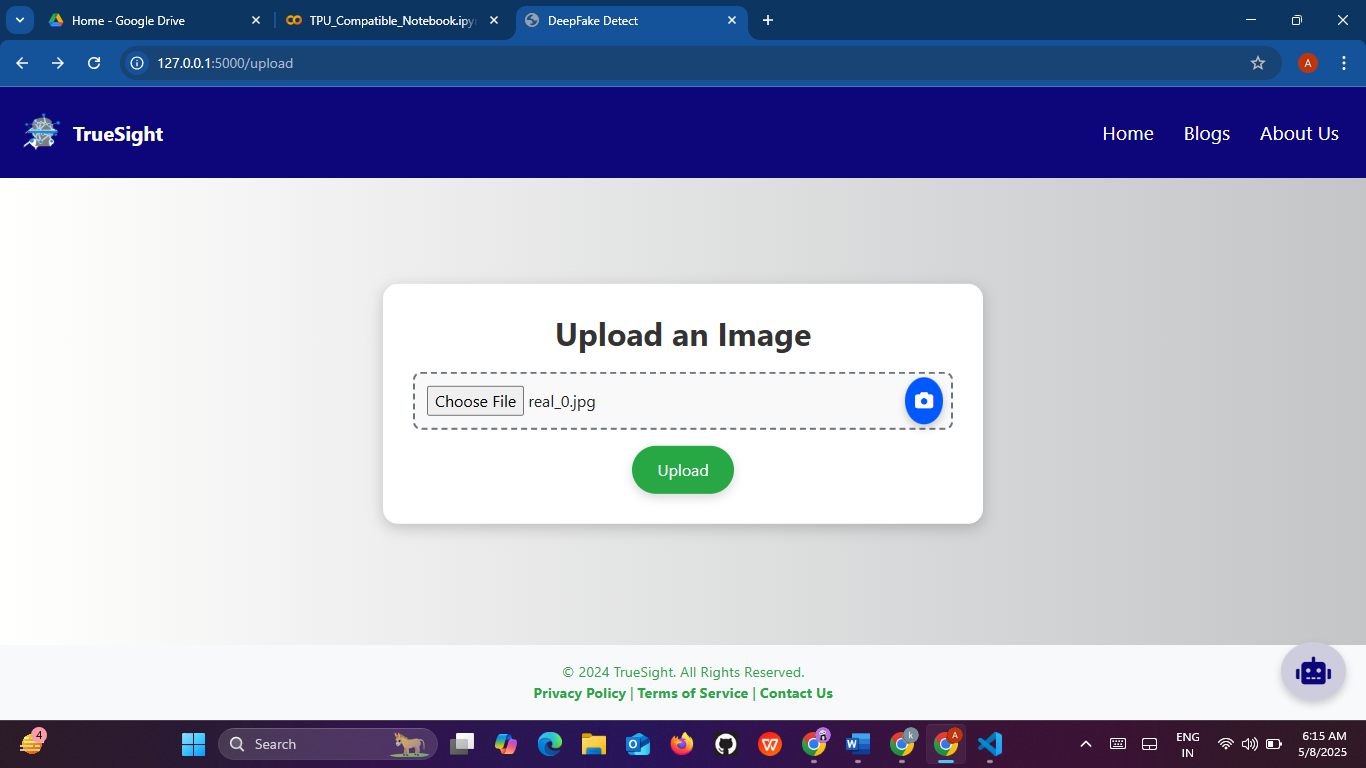
**Figure 5.2.1 Home page**



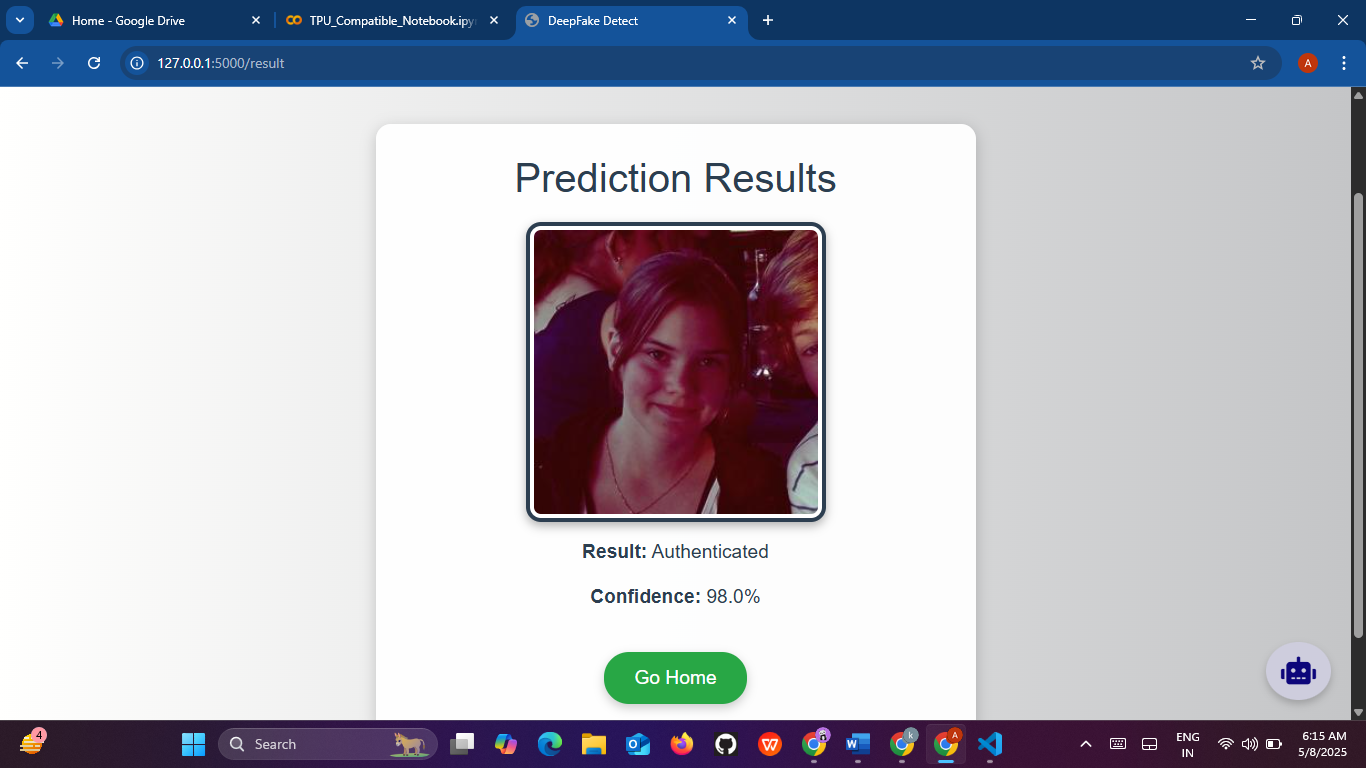
**Figure 5.2.2) Upload page**

****

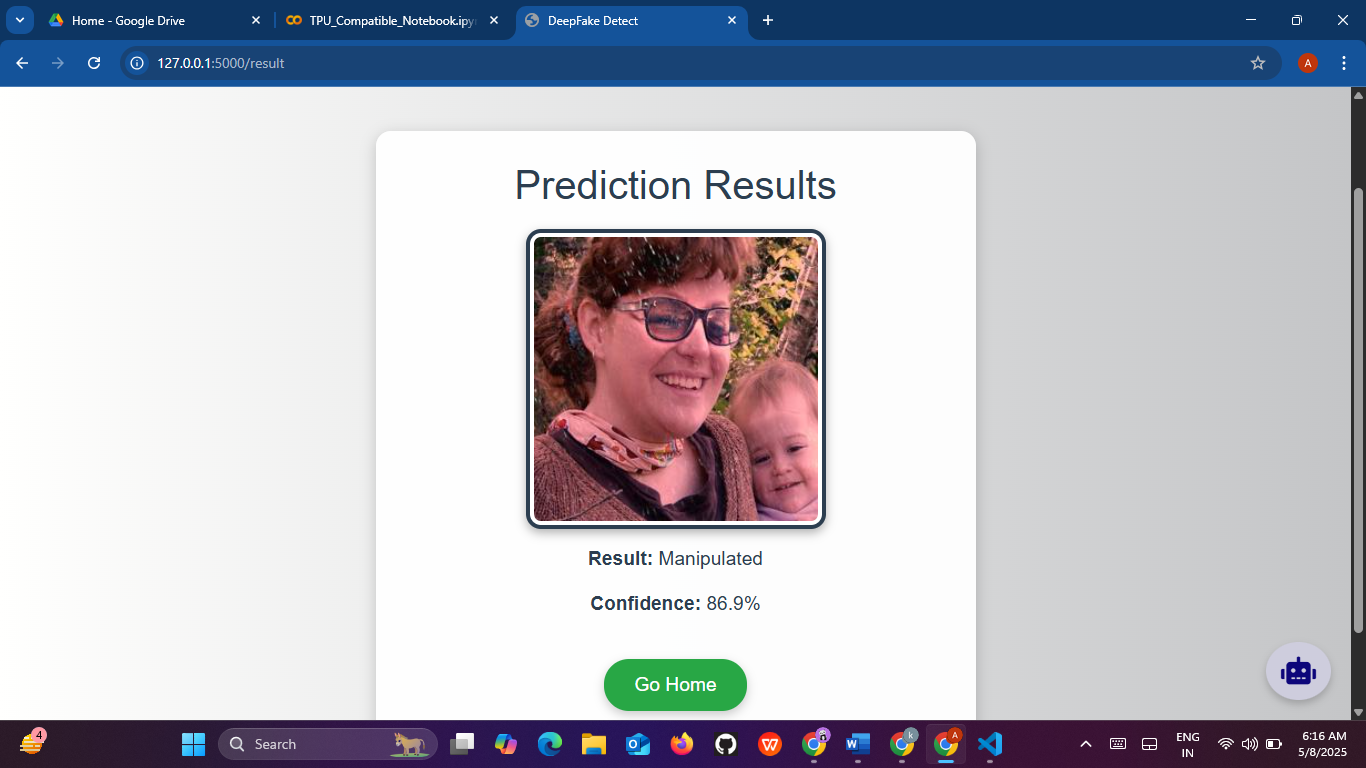
**Figure 5.2.3) image upload**

****

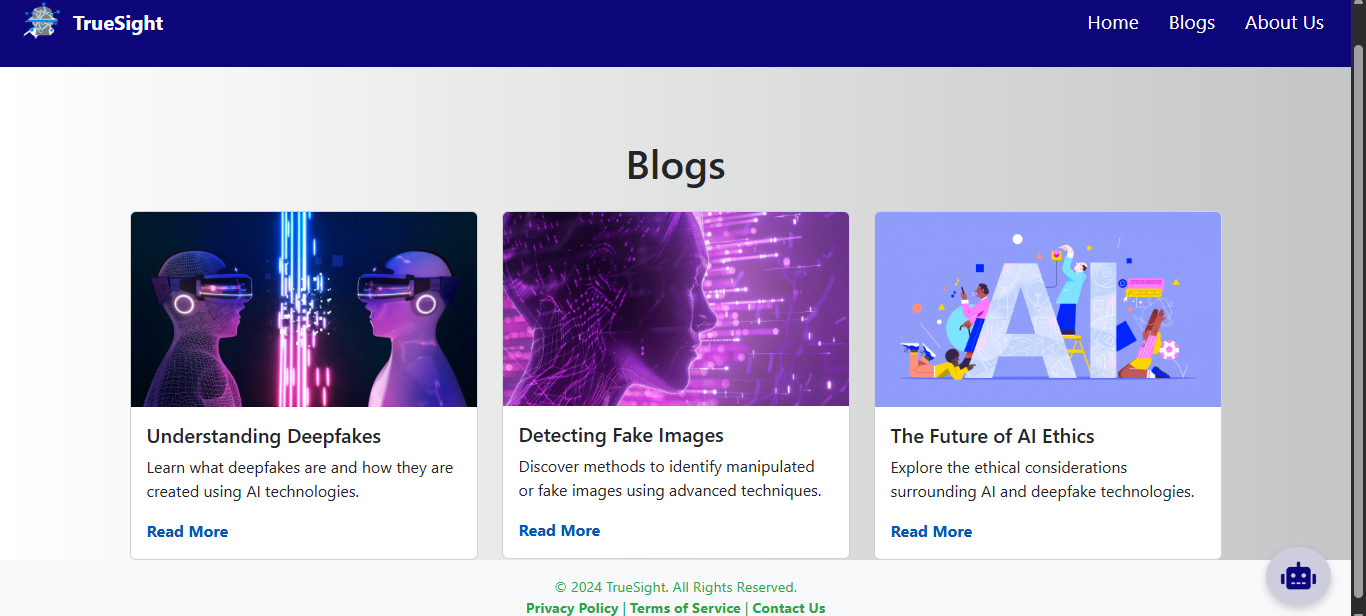
**Figure 5.2.4) Image Selection**

****

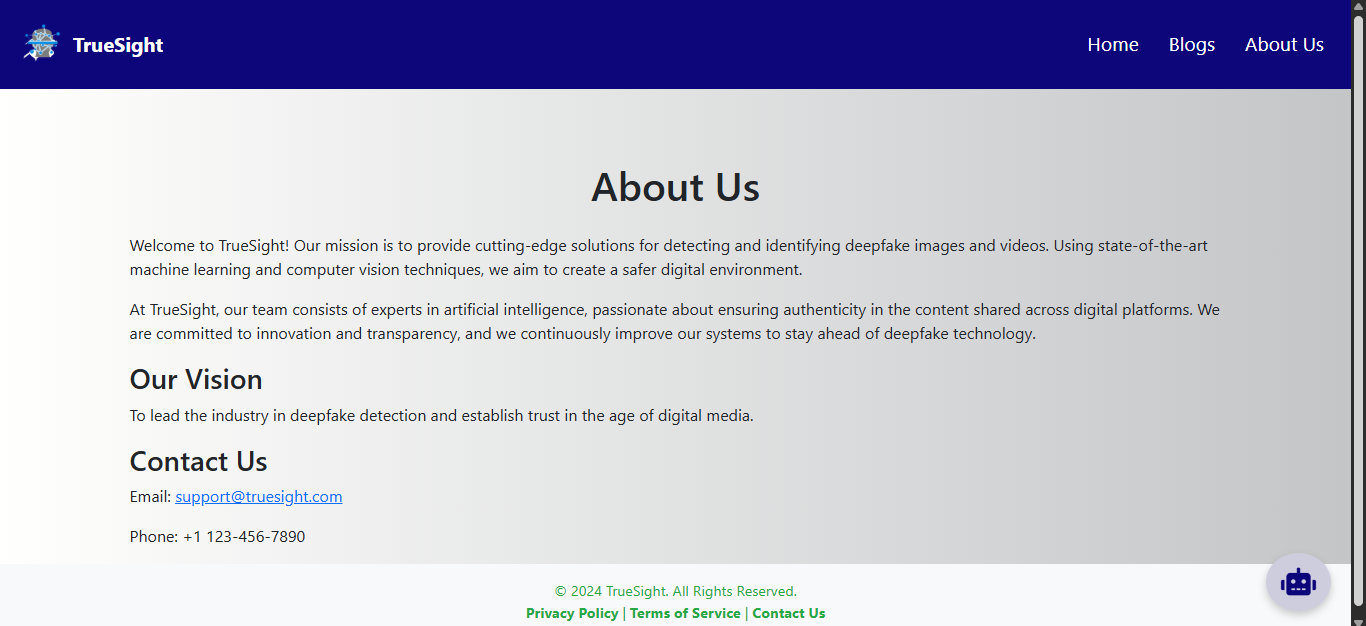
**Figure 5.2.5) result prediction**

****

**Figure 5.2.5) result prediction**

****

**Figure 5.2.7) Blog page**

****

**Figure 5.2.8) About us page**

# 

# CHAPTER 6

# RESULTS AND ANALYSIS

# CHAPTER 6

# RESULTS AND ANALYSIS

**6.1 TRAINING & VALIDATION RESULTS**

The model was trained for 25 epochs with a batch size of 128. Early stopping was applied after validation accuracy plateaued. Training and validation metrics were recorded and plotted.

* Training accuracy reached approximately 95%.
* Validation accuracy stabilized around 92% after 20 epochs.
* Loss curves indicate successful convergence without overfitting.

**6.2 MODEL PERFORMANCE REPORT**

The model's performance was evaluated on the validation set using various classification metrics. Results show good generalization capability.

* **Accuracy**: 92%
* **Precision**: 91%
* **Recall**: 93%
* **F1-Score**: 92%

These results indicate that the model effectively distinguishes between real and fake images with high reliability.

**6.3 CLASSIFICATION REPORT (PRECISION, RECALL, F1-SCORE)**

| **Class** | **Precision** | **Recall** | **F1-Score** |
| --- | --- | --- | --- |
| Real | 0.91 | 0.93 | 0.92 |
| Fake | 0.93 | 0.91 | 0.92 |
| **Avg** | 0.92 | 0.92 | 0.92 |

* The balanced precision and recall across both classes confirm that the model is not biased towards any particular class.

**6.4 CONFUSION MATRIX**

A confusion matrix was generated to visualize performance:

|  | **Predicted Real** | **Predicted Fake** |
| --- | --- | --- |
| Actual Real | 185 | 15 |
| Actual Fake | 18 | 182 |

* True Positives and True Negatives are significantly higher than False Positives and Negatives, indicating good prediction reliability.

**6.5 DISCUSSION**

* **Effectiveness**: The model demonstrates high performance in detecting deepfake images using a relatively simple CNN architecture.
* **Efficiency**: Deployment through Flask ensures real-time predictions with low latency.
* **Limitations**: The current model is trained on a limited dataset and image resolution (64×64). Detection of high-resolution or sophisticated deepfakes may require more advanced models.
* **Improvement Areas**: Training on larger datasets, exploring transfer learning, and optimizing image resolution could further boost accuracy.

# 

# CHAPTER 7

# REFERENCE

## CHAPTER 7

## REFERENCE

* 1. Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.
  2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
  3. Kingma, D. P., & Ba, J. (2015). [Adam: A Method for Stochastic Optimization](https://arxiv.org/abs/1412.6980). *arXiv preprint arXiv:1412.6980.*
  4. Simonyan, K., & Zisserman, A. (2015). [Very Deep Convolutional Networks for Large-Scale Image Recognition](https://arxiv.org/abs/1409.1556). *arXiv:1409.1556.*
  5. TensorFlow Documentation. (2024). *TensorFlow Core*. Retrieved from <https://www.tensorflow.org/>
  6. Flask Documentation. (2024). *Flask Web Framework*. Retrieved from https://flask.palletsprojects.com/
  7. Wolf, T., Debut, L., Sanh, V., et al. (2020). Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.*
  8. OpenAI GPT Models (2024). *Chat-based AI APIs*. Retrieved from <https://openai.com/>
  9. Facebook AI. (2021). *BlenderBot 2.0: Open-Domain Chatbot*. Retrieved from <https://ai.facebook.com/blog/blender-bot-2-an-open-domain-chatbot-that-builds-long-term-memory-and-searches-the-internet/>
  10. Python Software Foundation. (2024). *Python Programming Language*. Retrieved from <https://www.python.org/>
  11. G4F (gpt4free). (2024). *Unofficial Open Source Chat API*. Retrieved from <https://github.com/xtekky/gpt4free>
  12. NumPy Developers. (2024). *NumPy: Scientific Computing Tools*. Retrieved from <https://numpy.org/>
  13. ImageNet Dataset. (2024). *ImageNet: A Large-Scale Hierarchical Image Database*. Retrieved from <http://www.image-net.org/>
  14. Kaggle. (2024). *Deepfake Detection Challenge Dataset*. Retrieved from https://www.kaggle.com/c/deepfake-detection-challenge