DEEP LEARNING BASE FACE MORPHING DETECTION

Submitted in partial fulfilment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering
with Specialization in Artificial Intelligence and Machine Learning
by

G SIVA NAGA SAI (REG NO - 41611060)
P JASHWANTH (REG NO - 41733057)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Category – 1 University by UGC
Accredited with Grade "A++" by NAAC| Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119
April-2025

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Category - I University by UGC
Accredited "A++" by NAAC | Approved by AICTE
www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of **P JASHWANTH (REG NO: 41733057)** and **G SIVA NAGA SAI (REG NO: 41611060)**, who carried out Project entitled **"DEEP LEARNING BASE FACE MORPHING DETECTION"** under my supervision from November 2024 to April 2025.

Internal Guide

Dr. USAMA ABDUR RAHMAN, M.Tech., Ph.D.,

Head of the Department

Dr. S. VIGNESHWARI, M.E., Ph.D.

Submitted for Viva voce Examination held on 05/4/2025

Internal Examiner

External Examiner

DECLARATION

I, **G SIVA NAGA SAI (REG NO: 41611060)**, hereby declare that the Project Report entitled **"DEEP LEARNING BASE FACE MORPHING DETECTION"** done by me under the guidance of **Dr. USAMA ABDUR RAHMAN, M.Tech., Ph.D.,** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering in Artificial Intelligence and Machine Learning**

DATE: 05/04/2025

PLACE: CHENNAI

G. Siva Naga Sai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **Sathyabama Institute of Science and Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph. D.**, **Dean**, School of Computing, and **Dr. S. VIGNESHWARI, M.E., Ph.D., Head of the Department** of Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. USAMA ABDUR RAHMAN, M.Tech., Ph.D.,** for her valuable guidance, suggestions, and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering in Artificial Intelligence and Machine Learning** who were helpful in many ways for the completion of the project.

ABSTRACT

Deep learning-based face morphing detection has emerged as a critical tool for enhancing identity verification systems, which are vulnerable to sophisticated attacks using morphed facial images. This abstract explores the significance of employing deep learning techniques to detect face morphing, ensuring the integrity and reliability of identity verification processes. Face morphing involves blending facial features from multiple images to create a single, synthetic identity that can deceive conventional verification methods. Such attacks exploit the limitations of existing systems, which often rely on simple biometric checks. Deep learning models, particularly convolutional neural networks (CNNs), have shown remarkable capability in discerning subtle differences between authentic and morphed faces by analyzing pixel-level details and structural inconsistencies. This study reviews various deep learning architectures and methodologies employed for face morphing detection, highlighting their effectiveness in differentiating between genuine and manipulated facial images. Techniques such as feature extraction, similarity metrics, and adversarial training play pivotal roles in enhancing the robustness of detection systems against evolving morphing techniques. In the rapidly advancing field of biometric security, face morphing detection has emerged as a critical tool to fortify identity verification systems against sophisticated attacks. The increasing sophistication of identity fraud through face morphing attacks has underscored the urgent need for advanced detection systems in biometric security.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviations		Description
AI	-	Artificial Intelligence
CNN	-	Convolutional Neural Network
MFCC	-	Mel-Frequency Cepstral Coefficients
DL	-	Deep Learning
ML	-	Machine Learning
RNN	-	Recurrent Neural Network
GAN	-	Generative Adversarial Network
DNN	-	Deep Neural Network
MTCNN	-	Multi-Task Cascaded Convolutional Neural Network
ResNet	-	Residual Network
SVM	-	Support Vector Machine

CHAPTER 1

INTRODUCTION

As digital security measures become increasingly vital in our interconnected world, identity verification systems have become a cornerstone for ensuring secure access across a wide range of applications, from border control to financial services. Among the various biometric methods employed for identity verification, facial recognition has gained significant traction due to its convenience and non-intrusive nature. However, with the rise of advanced image manipulation techniques, these systems are facing new and sophisticated threats, particularly through the use of face morphing. Face morphing is a technique that blends the facial features of two or more individuals to create a synthetic image that can appear authentic enough to fool traditional verification systems. This manipulation exploits the limitations of current biometric checks, which often fail to detect such subtle alterations, leading to potential security breaches. The growing sophistication of these attacks necessitates the development of more advanced detection mechanisms to ensure the reliability and integrity of identity verification processes.

In this context, deep learning has emerged as a powerful tool to combat face morphing attacks. Leveraging the strengths of convolutional neural networks (CNNs) and other deep learning models, researchers are now able to detect even the most subtle differences between real and morphed faces by analyzing intricate pixel-level details and identifying structural inconsistencies that are typically imperceptible to the human eye. Face morphing attacks have been made increasingly feasible by the advent of deep learning techniques, such as Generative Adversarial Networks (GANs), which are capable of generating highly realistic facial images. These advancements in image synthesis, combined with the availability of large, diverse datasets, have lowered the barrier to entry for attackers, enabling them to create morphed images that are difficult to distinguish from authentic ones. As a result, the traditional methods of biometric verification—relying on simple distance metrics, geometric feature matching, or linear classifiers—are increasingly

ineffective at preventing these sophisticated attacks. To counter these emerging threats, researchers have turned to more advanced and sophisticated methods, particularly deep learning models, which have shown considerable promise in detecting face morphing.

To further enhance the effectiveness of face morphing detection, researchers are exploring a combination of deep learning techniques and traditional forensic analysis. Feature extraction methods, such as Local Binary Patterns (LBP) and Histogram of Oriented Gradients (HOG), can be integrated with deep neural networks to improve accuracy and robustness. Additionally, explainable AI (XAI) approaches are being incorporated to provide greater transparency into model decisions, enabling security professionals to better understand the factors influencing morphing detection. Moreover, advancements in adversarial training and data augmentation techniques help models generalize better across different morphing methods and variations in image quality. Future research in this field is focused on developing lightweight and efficient models that can be deployed in real-time identity verification systems, ensuring scalability across various applications, including mobile authentication, border security, and financial transactions. By continuously refining detection methods and leveraging the latest advancements in AI and machine learning, the fight against face morphing attacks can stay ahead of evolving threats, safeguarding the integrity of biometric verification systems.

This hybrid approach mitigates the risk posed by single-modality vulnerabilities and enhances overall system robustness. Additionally, the development of large-scale benchmark datasets specifically designed for face morphing detection is playing a crucial role in training and evaluating deep learning models, enabling researchers to refine detection algorithms and improve generalization across diverse real-world scenarios. As face morphing techniques continue to evolve, leveraging emerging AI-driven countermeasures, such as transformer-based models and self-supervised learning, can further enhance detection capabilities.

CHAPTER 2

LITERATURE SURVEY

2.1 INFERENCES FROM LITERATURE SURVEY

A. Maithresh, V. Nikhil, H. Saipuneeth and G. V. S. Reddy, "Exploring the Superiority of CapsNet over CNN For Early Detection of Lung Cancer A Comparative Inventive Computation Technologies (ICICT), Lalitpur, Nepal, 2023. [1] This study presents a comparative analysis of Capsule Networks (CapsNet) and Convolutional Neural Networks (CNN) in detecting early-stage lung cancer using medical imaging techniques. The authors highlight the advantages of CapsNet in preserving spatial hierarchies, improving feature representation, and reducing misclassifications, especially in small datasets. Their experimental results demonstrate that CapsNet outperforms traditional CNNs in terms of accuracy, robustness, and generalization capability. The paper also discusses the challenges of deep learning in medical imaging, including data scarcity, computational complexity, and the need for explainability in AI-driven diagnostic systems.

D. Gharde, M. P. A, S. N and S. K. S, “Detection of Morphed Face, Body, Audio signals using Deep Neural Networks,” 2022. [2] This study explores the application of deep neural networks (DNNs) for detecting morphed facial images, body manipulations, and synthetic audio signals. The authors emphasize the growing threat posed by advanced image and voice synthesis techniques, including Generative Adversarial Networks (GANs) and deepfake technologies, which make it increasingly difficult to distinguish between real and manipulated content. The research examines various deep learning architectures, such as Convolutional Neural Networks (CNNs) for image forensics and Recurrent Neural Networks (RNNs) for detecting synthetic speech patterns. Additionally, the paper highlights the role of feature extraction techniques like Mel-Frequency Cepstral Coefficients (MFCC) for audio analysis and structural consistency

metrics for face morphing detection.

D. K. S and M. T. I, "Deep learning architectures for Brain Tumor detection: A Survey," 2023 Advanced Computing and Communication Technologies for High Performance Applications. [3] This survey provides a comprehensive review of deep learning architectures used for brain tumor detection, highlighting the advancements and challenges in medical image analysis. The authors discuss various neural network models, including Convolutional Neural Networks (CNNs), Residual Networks (ResNets), Vision Transformers (ViTs), and Generative Adversarial Networks (GANs), emphasizing their role in improving tumor classification and segmentation accuracy. The paper examines multiple benchmark datasets, such as BraTS (Brain Tumor Segmentation) and Kaggle's brain MRI datasets, to evaluate the performance of different deep learning approaches. The study also explores the integration of Explainable AI (XAI) techniques to enhance model interpretability and aid medical professionals in decision-making. Additionally, the authors discuss challenges such as data scarcity, overfitting, and the computational demands of deep learning models, proposing potential solutions to optimize performance in real-world clinical settings.

F. Mokhayeri, E. Granger and G. -A. Bilodeau, "Synthetic face generation under various operational conditions in video surveillance," 2015. [4] This study investigates the generation of synthetic facial images under diverse operational conditions in video surveillance systems. The authors explore the impact of varying environmental factors such as lighting conditions, occlusions, motion blur, and camera angles on synthetic face generation and recognition. The research leverages advanced image synthesis techniques, including Generative Adversarial Networks (GANs) and traditional augmentation methods, to create high-fidelity synthetic faces for training and testing surveillance models. Additionally, the study examines the role of deep learning-based facial recognition systems in handling synthetic face variations and their effectiveness in improving identity verification in real-time surveillance environments.

He, Z. Xu, Y. Chen, Z. Pan and I. Chih-Lin, "Work in progress paper: Network

deployment and operation based on spatial and temporal traffic model," 2014. [5] This work-in-progress paper presents a novel approach to network deployment and operation, leveraging spatial and temporal traffic models to optimize performance. The authors explore how real-world traffic patterns, including user mobility, peak-hour fluctuations, and geographic distribution, influence network resource allocation and infrastructure planning. The study incorporates machine learning techniques and predictive analytics to enhance network adaptability, enabling dynamic adjustments to bandwidth, latency, and service quality based on demand variations. Additionally, the paper discusses the challenges of real-time network optimization in 5G and beyond, focusing on scalability, cost-efficiency, and energy consumption. Preliminary findings suggest that integrating spatial-temporal traffic models can significantly enhance network reliability and efficiency, reducing congestion and improving overall user experience.

M. -H. Kim, D. -S. Wang, S. -T. Wang, S. -H Park and C. -G. Lee, "Improving the Robustness of the Bug Triage Model through Adversarial Training," 2022. [6] This paper explores methods to enhance the robustness of bug triage models using adversarial training techniques. Bug triage, a critical process in software maintenance, involves categorizing and assigning software defects to the appropriate development teams. Traditional machine learning-based triage models often struggle with noisy and adversarial inputs, which can degrade classification accuracy and increase misassignments. The authors propose an adversarial training approach, where perturbed versions of bug reports are introduced during model training to improve resilience against misleading or ambiguous inputs. The study examines various adversarial attack scenarios, such as mislabeled bug reports, duplicate issues, and textual inconsistencies, to assess their impact on triage performance.

Q. Zhang and H. He, "Research of an Enhanced YOLOv5-Based Algorithm for Garbage Detection in Service Areas," 2024. [7] This study presents an improved garbage detection system using an enhanced version of the YOLOv5 (You Only Look Once v5) object detection algorithm. The authors focus on optimizing garbage identification in service areas, such as public spaces, transportation

hubs, and commercial zones, where efficient waste management is crucial. The proposed enhancements include fine-tuning YOLOv5's feature extraction layers, incorporating attention mechanisms, and integrating image preprocessing techniques to improve detection accuracy in complex environments. The study also explores real-world challenges such as occlusions, varying lighting conditions, and diverse waste categories. Experimental results demonstrate that the improved YOLOv5 model outperforms standard detection frameworks in terms of precision, recall, and inference speed. Additionally, the authors discuss the potential applications of this model in smart city waste management, automated cleaning systems, and environmental monitoring.

S. K. Dubey, S. Gupta, S. K. Pandey and S. Mittal, "Ethical Considerations in Data Mining and Database Research" 2024. [8] This paper explores the ethical implications of data mining and database research, emphasizing the balance between technological advancements and responsible data usage. As data mining techniques become increasingly sophisticated, concerns related to privacy, bias, consent, and data security have gained significant attention. The authors discuss key ethical challenges, including the risk of unauthorized data access, algorithmic discrimination, and the potential misuse of personal information. The study also highlights the importance of regulatory frameworks such as the General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) in guiding ethical data practices. Additionally, the paper examines techniques for mitigating ethical risks, including differential privacy, federated learning, and transparency in AI-driven decision-making. Experimental case studies illustrate the consequences of unethical data mining practices, reinforcing the need for accountability and fairness in database research.

V. Gunturu, N. Maiti, B. Toure, P. Kunekar, S. B. Banu and D. Sahaya Lenin, "Transfer Learning in Biomedical Image Classification," 2024 [9] This paper explores the application of transfer learning in biomedical image classification, highlighting its effectiveness in improving model performance with limited labeled medical datasets. Traditional deep learning models require vast amounts of annotated data to achieve high accuracy, which is often a challenge in the

biomedical domain due to privacy concerns, data scarcity, and annotation costs. The authors review various pre-trained deep learning architectures, including ResNet, VGG, EfficientNet, and DenseNet, and their adaptation for tasks such as tumor detection, disease classification, and anomaly identification in medical imaging modalities like MRI, CT scans, and X-rays.

X. Chen, Y. Zou and H. Ke, "TrafficYOLO: YOLO with Multi-Head Attention Mechanism for Traffic Detection Scenarios," [10] This paper presents TrafficYOLO, an enhanced version of the YOLO **(**You Only Look Once**)** object detection model, specifically designed for traffic monitoring and detection applications. The proposed model integrates a Multi-Head Attention Mechanism (MHAM) to improve the detection accuracy of vehicles, pedestrians, and road signs in dynamic and complex traffic environments. The authors address key challenges in traffic detection, including occlusions, varying lighting conditions, motion blur, and real-time processing constraints. Additionally, the paper discusses the deployment of TrafficYOLO on edge devices, highlighting its efficiency in low-power environments for real-time traffic surveillance, smart city applications, and autonomous driving systems. Future research directions include further optimizing the attention mechanism, integrating sensor fusion techniques (e.g., LiDAR and radar data), and exploring self-supervised learning approaches to enhance model robustness.

2.2 EXISTING SYSTEM AND PROPOSED SYSTEM

2.2.1 *EXISTING SYSTEM*

Traditional face verification systems have long relied on biometric authentication techniques that use facial recognition to match an individual's image against a stored template. These systems are commonly used in security applications such as passport verification, border control, financial transactions, and access control systems. The core methodologies include feature extraction, landmark detection, and similarity matching, where the system compares key facial features, such as the distance between the eyes, nose shape, and jawline structure, to authenticate an individual. However, with the rise of advanced face morphing techniques, these conventional methods are increasingly vulnerable to attacks where a morphed

image can pass as a valid identity for multiple individuals.

o combat this, existing detection approaches employ handcrafted feature extraction methods, such as Local Binary Patterns (LBP), Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), and Discrete Cosine Transform (DCT), which analyze texture variations, symmetry, and edge patterns in an image. These methods attempt to identify inconsistencies introduced by morphing, such as unnatural blending of facial features or texture artifacts. Additionally, machine learning-based classifiers like Support Vector Machines (SVMs), Random Forests, and k-Nearest Neighbors (k-NN) have been used to train models for morphing detection. While these approaches offer moderate success in controlled conditions, they struggle with high-quality, AI-generated morphs that exhibit minimal visual distortions.

Another major challenge with the existing systems is their inability to generalize across different datasets and real-world scenarios. Many biometric verification systems are designed with static feature extraction rules, making them susceptible to sophisticated morphing techniques generated by Generative Adversarial Networks (GANs), Autoencoders, and Deepfake-based synthesis methods. Such AI-driven face morphing techniques create images that retain high perceptual realism, bypassing traditional detection mechanisms. As a result, standard biometric authentication systems fail to differentiate between real and morphed identities, leading to security vulnerabilities.

Additionally, most current systems do not incorporate real-time detection capabilities or large-scale datasets that reflect diverse lighting conditions, facial expressions, and ethnic variations. This lack of robustness significantly reduces their effectiveness in real-world applications. Moreover, adversarial attacks can manipulate input images to deceive face recognition models, making it even harder for existing methods to maintain accuracy. To address these limitations, modern face morphing detection systems are shifting towards deep learning-based approaches, leveraging Convolutional Neural Networks (CNNs), attention

mechanisms, and adversarial training to enhance detection accuracy and security.

Disadvantages of the Existing System:

- Low Detection Accuracy – Traditional methods struggle to detect morphs created using advanced techniques like Generative Adversarial Networks (GANs), Autoencoders, and Deepfake-based synthesis.
- Poor Generalization – Existing systems fail under varying lighting conditions, facial expressions, and diverse ethnic backgrounds, leading to inconsistent results.
- High False Acceptance Rate (FAR) – Morphing attacks can bypass biometric verification, allowing unauthorized access and posing identity fraud risks.
- Lack of Real-Time Processing – Most methods require extensive computational time, making them unsuitable for fast authentication in security-sensitive applications like border control.
- Vulnerability to Adversarial Attacks – Small, imperceptible modifications can easily deceive traditional models, making them ineffective against sophisticated cyber threats.

2.2.2 PROPOSED SYSTEM

To address the limitations of traditional face morphing detection methods, the proposed system leverages deep learning-based techniques, particularly Convolutional Neural Networks (CNNs), attention mechanisms, and adversarial training, to enhance detection accuracy and robustness. This approach ensures reliable identity verification by effectively distinguishing between authentic and morphed facial images. It also employs data augmentation and synthetic morph generation to improve model generalization across diverse datasets and real-world conditions. To further enhance security, the system integrates multi-layer anomaly detection algorithms, which analyze spatial and temporal inconsistencies in facial structures. Finally, the proposed approach ensures seamless integration with existing biometric verification frameworks, making it adaptable for large-scale deployment in border security, banking, and national ID verification systems.

Deep Learning-Based Facial Feature Extraction

The proposed system employs Convolutional Neural Networks (CNNs) to extract deep facial features that are otherwise imperceptible to human eyes. Unlike traditional handcrafted feature extraction techniques, CNNs can capture complex spatial relationships and pixel-level anomalies in facial images. By utilizing pre-trained architectures such as ResNet, EfficientNet, and MobileNet, the system ensures accurate and efficient morphing detection. The feature maps generated by these deep networks allow the system to distinguish subtle artifacts introduced during morphing, thereby improving detection accuracy against high-quality, AI-generated morphs.

Attention Mechanisms for Enhanced Detection

The system integrates transformers and attention-based models, such as the Vision Transformer (ViT), to enhance the detection of fine-grained distortions in morphed images. Attention mechanisms help focus on the most significant areas of an image by dynamically adjusting weights to regions with morphing artifacts, texture inconsistencies, and unnatural feature blending. This approach allows for better localization of manipulated regions, thereby reducing false positives and false negatives. By leveraging multi-head self-attention, the system can process global dependencies within the image, leading to superior morphing detection performance.

Adversarial Training to Counter Deepfake Attacks

The system incorporates adversarial training using Generative Adversarial Networks (GANs) to strengthen resistance against intelligent morphing techniques and deepfake manipulations. GAN-based adversarial augmentation generates challenging morphed images, which the detection model is trained to recognize, ensuring its adaptability to evolving attack methods. This technique also enhances the model's robustness by exposing it to adversarial noise and synthetic variations, making it highly resilient to manipulated face images that bypass traditional security checks.

□

Multi-Modal Biometric Integration

To further increase security, the proposed system integrates multiple biometric modalities, such as eye movement analysis, skin texture detection, and facial depth estimation. Traditional biometric systems often rely solely on facial image verification, which makes them vulnerable to face morphing attacks. By incorporating additional biometric markers, the system ensures a multi-layered authentication approach, significantly reducing the chances of morphed images being falsely accepted. This combination of modalities enhances the overall accuracy and robustness of identity verification systems.

Explainable AI (XAI) for Decision Transparency

One major limitation of deep learning models is their black-box nature, making it difficult to understand why a particular decision was made. The proposed system incorporates Explainable AI (XAI) techniques, such as Grad-CAM (Gradient-weighted Class Activation Mapping) and SHAP (Shapley Additive Explanations), to provide visual explanations for detected morphing artifacts. This feature is particularly beneficial for forensic experts and security analysts, as it helps them validate and interpret model predictions, making the detection process more transparent and trustworthy.

Real-Time Face Morphing Detection

The proposed system is optimized for real-time processing, ensuring it can function efficiently in high-security environments like border control, financial transactions, and airport security checkpoints. Through efficient deep learning models and hardware-optimized implementations using GPU acceleration and edge AI, the system can analyze and verify facial images within milliseconds. The use of lightweight models like MobileNet and TensorRT optimizations makes it feasible to deploy the system on cloud servers, mobile devices, and embedded systems, allowing seamless integration into existing security frameworks.

Scalability and Cloud-Based Deployment

The system is designed for scalability, allowing deployment in large-scale security

infrastructures such as national identity verification systems, immigration control, and online banking authentication. Using containerized solutions like Docker and Kubernetes, the system can efficiently handle high volumes of face verification requests with minimal computational overhead. Additionally, cloud integration with platforms like AWS, Azure, or Google Cloud ensures high availability and seamless updates, making it adaptable to future advancements in face morphing techniques.

2.3 OPEN PROBLEMS IN EXISTING SYSTEM

Despite advancements in face morphing detection, existing systems still struggle with several unresolved challenges that compromise their reliability and security. One of the most pressing issues is low detection accuracy against high-quality morphs generated using deep learning-based techniques like Generative Adversarial Networks (GANs) and Deepfake models. These sophisticated algorithms produce morphed images with seamless blending, making it extremely difficult for traditional biometric verification systems to differentiate between real and manipulated faces. Additionally, existing detection models lack generalization across diverse datasets, as they often fail to perform effectively under varying lighting conditions, facial expressions, and different demographic groups. This inconsistency limits their applicability in real-world security applications, such as border control and financial authentication systems.

2.3.1 Limited Generalization Across Diverse Datasets

Most face morphing detection systems are trained on predefined datasets, which may lack ethnic diversity, varied lighting conditions, and real-world variations found in different sources like passport images, social media uploads, and surveillance footage. As a result, models trained on specific datasets often fail to generalize effectively when tested on unseen data, leading to biased predictions and decreased accuracy. Additionally, the lack of standardized datasets for benchmarking face morphing detection algorithms makes it difficult to compare models fairly and improve their robustness across different environments.

2.3.2 High False Acceptance and Rejection Rates

Many existing face morphing detection systems suffer from imbalanced false acceptance rates (FAR) and false rejection rates (FRR), making them unreliable in practical applications. False acceptance occurs when a morphed image is incorrectly classified as real, allowing unauthorized access, while false rejection mistakenly flags genuine images as morphed, causing unnecessary security blocks. These issues arise due to inefficient feature extraction techniques, inadequate threshold settings, and poor adaptability to different face morphing methods. In high-security applications like border control, banking, and national identity verification, such inconsistencies can lead to identity fraud risks, security loopholes, and operational inefficiencies.

2.3.3. Lack of Real-Time Detection and High Computational Cost

Many existing face morphing detection approaches rely on complex feature extraction techniques and computationally expensive models, making them slow and impractical for real-time applications. Traditional methods often require high-resolution image processing, multi-stage classification pipelines, and exhaustive comparison with reference datasets, resulting in delays in decision-making. This is especially problematic for applications like airport security, financial transactions, and surveillance systems, where quick identity verification is crucial.

2.3.4. Inability to Detect Advanced AI-Generated Morphs

The rapid advancements in Generative Adversarial Networks (GANs), Autoencoders, and Neural Style Transfer techniques have enabled attackers to create highly realistic synthetic facial images that bypass traditional detection mechanisms. Conventional face morphing detection techniques, which rely on handcrafted features, local binary patterns (LBP), and statistical texture analysis, struggle to identify these deep-learning-generated morphs due to their high visual fidelity. Modern morphing attacks can produce seamless blending of facial features, making it nearly impossible for older detection systems to distinguish between real and manipulated images. This lack of robustness leaves identity verification systems vulnerable to AI-powered deception, increasing the risk of unauthorized access and fraud.

2.3.5. *Vulnerability to Adversarial Attacks and Spoofing Techniques*

Many face morphing detection models are highly susceptible to adversarial attacks, where attackers introduce subtle perturbations to facial images that are imperceptible to the human eye but can fool machine learning models into misclassifications. Adversarial examples can be created using gradient-based optimization techniques, which alter the pixel distribution in a way that disrupts traditional classification models. Additionally, existing systems lack robustness against common spoofing techniques, such as using printed morphed images, video replays, or 3D mask reconstructions to deceive biometric authentication systems. Since most detection frameworks primarily focus on static image analysis, they fail to incorporate multi-modal biometric cues, such as facial depth maps, liveness detection, or micro-expression tracking, which are essential to prevent sophisticated morphing-based identity fraud.

2.3.6. Limited Explainability and Transparency in Decision-Making

One of the major drawbacks of existing deep learning-based face morphing detection systems is their black-box nature, meaning they provide predictions without offering clear explanations for their decisions. In security-critical applications, human operators, forensic analysts, and law enforcement agencies require interpretable and explainable models to validate the detection results before taking action. However, most current models lack explainable AI (XAI) techniques, such as Grad-CAM (Gradient-weighted Class Activation Mapping) or SHAP (Shapley Additive Explanations), which could highlight the specific facial regions responsible for the classification. Without such insights, it becomes challenging to trust the model’s decisions, and legal systems may not accept AI-generated evidence for authentication disputes.

CHAPTER 3

AIM AND SCOPE OF PRESENT INVESTIGATION

3.1 AIM

The primary aim of this project is to develop a deep learning-based face morphing detection system that enhances the security and reliability of biometric identity verification. With the rise of advanced image manipulation techniques, including Generative Adversarial Networks (GANs) and deepfake technology, traditional verification systems have become vulnerable to morphing attacks that allow unauthorized individuals to gain access. This project focuses on leveraging Convolutional Neural Networks (CNNs) and other deep learning architectures to detect subtle inconsistencies in facial images that are imperceptible to the human eye. By analyzing pixel-level details, feature distortions, and structural irregularities, the proposed system aims to accurately differentiate between real and morphed faces, reducing false acceptance and rejection rates.

3.1 SCOPE

The scope of this project extends to various high-security applications, including border control, financial transactions, digital identity verification, and law enforcement. By integrating real-time detection capabilities, this system ensures quick and efficient identification of fraudulent images, reducing false acceptance rates in biometric systems. Furthermore, the project explores adversarial training techniques to enhance model robustness against evolving morphing attacks, ensuring long-term reliability. Additionally, future enhancements may include multi-modal biometric integration, real-time deployment on edge devices, and improved generalization across diverse datasets to strengthen biometric security worldwide. The project also aims to support multi-platform deployment, making it adaptable for mobile authentication systems, cloud-based verification, and on-premise security solutions.

3.3 PRODUCT FUNCTIONS

The product supports real-time processing, enabling fast and efficient detection for applications such as border security, financial authentication, and national ID verification. It integrates with existing biometric systems to enhance security while reducing false acceptance and rejection rates. Additionally, the system is designed to adapt to evolving morphing techniques, including GAN-generated images and deepfake manipulations, ensuring continuous improvement through adversarial training and dataset expansion. Future enhancements may include multi-modal biometric verification, edge computing deployment, and liveness detection to strengthen identity protection and fraud prevention. The system can be deployed on cloud, edge devices, or integrated with on-premise security frameworks, making it versatile for various authentication environments. Additionally, it provides detailed reports and analytics to help security professionals track and mitigate morphing threats effectively.

3.4 USE CASES AND CHARACTERISTICS

The face morphing detection system has diverse applications across border security, financial authentication, government ID verification, law enforcement, and digital identity protection. It enhances biometric security by preventing morphing-based identity fraud in passports, banking KYC processes, and national ID verification systems. By leveraging deep learning techniques like CNNs, the system analyzes structural inconsistencies and pixel-level distortions to differentiate between real and morphed faces. Designed for real-time processing, it seamlessly integrates with existing authentication systems, ensuring fast and accurate verification. Its scalability and adaptability allow deployment on cloud, edge devices, and mobile applications, making it highly versatile. Additionally, the system continuously improves through adversarial training and adaptive learning, ensuring it stays ahead of evolving GAN-based and deepfake morphing techniques. With multi-modal biometric support and an intuitive interface, it provides detailed reports and analytics, making it a reliable tool for forensic investigations, fraud prevention, and secure digital authentication.

3.5 OPERATING ENVIRONMENT

The face morphing detection system is designed to operate in various computational environments to ensure seamless integration and efficiency across different security applications. It can be deployed on cloud-based servers, on-premise security infrastructures, and edge computing devices for real-time biometric verification. The system is optimized to work with high-performance GPUs and CPUs to process large volumes of image data efficiently. For software compatibility, it supports deep learning frameworks like TensorFlow and PyTorch, along with OpenCV for image processing and biometric verification APIs for integration with existing security systems. It is designed to run on Windows, Linux, and cloud platforms like AWS, Azure, and Google Cloud. Additionally, it can be deployed on mobile and embedded systems to support biometric authentication in smartphones, border security checkpoints, and access control systems.

3.6 CONSTRAINTS

The face morphing detection system faces several constraints, including high computational requirements, as deep learning models like CNNs demand powerful GPUs or TPUs for efficient processing, making deployment on low-end devices challenging. The system heavily depends on high-quality image inputs, as low-resolution or distorted images can lead to false detections. Additionally, evolving morphing techniques, especially those powered by GANs and deepfake models, require continuous model retraining and dataset updates to maintain accuracy. Integration with existing biometric systems may present compatibility issues, particularly with legacy infrastructures that lack support for deep learning-based verification. Ensuring real-time detection in applications such as border control and financial authentication is another challenge, as maintaining low-latency processing without compromising accuracy is complex. Privacy concerns also arise, as handling sensitive biometric data necessitates strict security and compliance measures to prevent misuse. Lastly, the system must be scalable, as deploying it across large networks like airports or government databases requires substantial storage and computing resources, ensuring stable performance as the user base grows.

3.7 PLATFORM

The face morphing detection system is designed to be platform-independent, ensuring flexibility in deployment across various environments. It can be implemented on cloud platforms such as AWS, Microsoft Azure, and Google Cloud, allowing for scalable, high-performance processing. For on-premise deployments, it supports Windows and Linux-based servers, leveraging high-performance GPUs and CPUs for efficient deep learning model execution. For real-time applications, the system can be integrated with edge computing devices like Raspberry Pi, NVIDIA Jetson, or other embedded AI hardware, enabling biometric verification in border security, financial services, and surveillance systems. It is also compatible with mobile platforms, allowing for deployment on Android and iOS devices for remote authentication solutions.

3.8 USE CASE MODEL

The face morphing detection system operates through a structured use case model, where multiple actors interact with the system to ensure secure identity verification. The process begins when a user submits their facial image for authentication, typically in applications like border security, financial verification, or government ID issuance. The system then extracts key facial features using CNN-based deep learning models and analyzes the image for morphing inconsistencies. If the system detects any signs of manipulation, it triggers an alert to the verification authority, such as airport security or bank officials, who take further action. The system also maintains a database of detected morphing attempts, improving future detection accuracy. System administrators oversee the process, ensuring model updates and security compliance. The AI-powered morph detection module continuously learns from new threats, enhancing its effectiveness against GAN-generated deepfake morphs.

CHAPTER 4

EXPERIMENTAL OR MATERIALS AND METHODS; ALGORITHMS USED

4.1 SELECTED METHODOLOGY

The face morphing detection system follows a structured methodology that integrates deep learning techniques with biometric security to accurately identify manipulated facial images. The process begins with data collection and preprocessing, where a dataset of authentic and morphed facial images is gathered and refined through techniques like image normalization, resizing, and augmentation to improve model performance. Feature extraction is then performed using Convolutional Neural Networks (CNNs), leveraging architectures such as VGG, ResNet, and EfficientNet to capture fine-grained pixel-level inconsistencies in facial structures. The extracted features are analyzed using similarity metrics and statistical models to differentiate between genuine and morphed faces.

4.2.1 *Algorithms used:*

The face morphing detection system in this abstract primarily relies on deep learning algorithms, particularly Convolutional Neural Networks (CNNs), for feature extraction and classification. Additionally, several advanced machine learning and deep learning techniques enhance the detection process.

Convolutional Neural Networks (CNNs) – CNNs are the backbone of the system, analysing pixel-level details to detect structural inconsistencies in morphed faces. Popular architectures like VGG, ResNet, and EfficientNet can be used to improve accuracy.

Generative Adversarial Networks (GANs) – GANs play a dual role: first, they are used to generate realistic morphed images for training, and second, they help improve the detection model's robustness through adversarial training.

Support Vector Machines (SVMs) and Random Forest – Traditional machine learning models can serve as secondary classifiers, analyzing extracted features from deep learning models for additional verification.

Autoencoders and Variational Autoencoders (VAEs) – These unsupervised learning techniques can be used for anomaly detection, identifying discrepancies in morphed images compared to authentic faces.

4.2 FLOWCHART

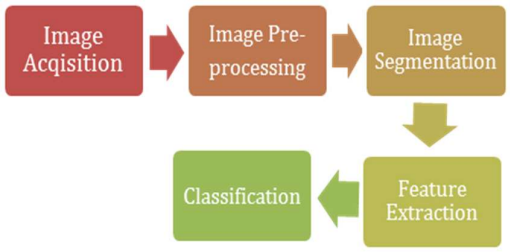

Fig 4.2 Work Flow

4.3 ARCHITECTURE OF PROPOSED SYSTEM

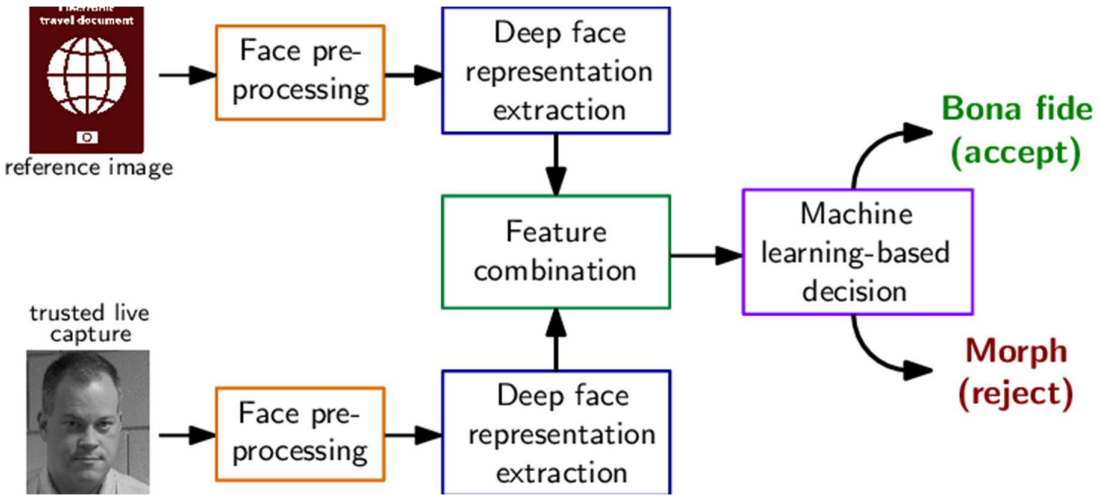

Fig 4.2 SYSTEM ARCHITECTURE

The given architecture represents a machine learning-based face morphing

detection system used for biometric identity verification. The process starts with two inputs: a reference image from a trusted document (such as a passport or ID) and a live capture of the individual's face. Both images undergo face pre-processing, which includes normalization, alignment, and noise reduction to enhance quality and ensure consistency. After pre-processing, a deep face representation extraction module is used to encode facial features into a high-dimensional feature space using deep learning models like CNNs or pre-trained face recognition networks. The extracted features from both images are then combined in the feature combination step, where similarities and discrepancies between the reference and live capture images are analyzed. This combined feature set is passed to a machine learning-based decision module, which classifies the input as either "Bona fide" (genuine face, accepted) or "Morph" (manipulated face, rejected).

4.3.1. Input Acquisition

The system begins by acquiring two types of facial images: a reference image and a live capture. The reference image is typically sourced from a secure document such as a passport, national ID, or driver’s license, ensuring its authenticity. The live capture is taken in real-time during the verification process, usually through a webcam or biometric scanning device. The purpose of collecting both images is to compare the live input against a trusted source to detect any morphing manipulations. This step is crucial in preventing identity fraud that exploits vulnerabilities in biometric authentication systems. Ensuring that the captured image meets quality standards is essential, as poor image quality may affect the system’s performance. Moreover, real-time validation of the captured image can help eliminate errors at the initial stage. A secure storage mechanism is required to prevent tampering with the reference image.

4.3.2. Face Pre-processing

Before feature extraction, both images undergo a series of pre-processing techniques to standardize their format and improve detection accuracy. This involves face detection, alignment, and normalization to ensure both images have similar orientations and lighting conditions. Additionally, noise reduction and artifact

removal techniques are applied to enhance image clarity by eliminating distortions caused by low-quality cameras or compression artifacts. The goal of this step is to create a uniform dataset where both images can be accurately compared without external variations affecting the results. Histogram equalization techniques may be used to enhance contrast and highlight facial features more effectively.

4.3.3. Deep Face Representation Extraction

Once the images are pre-processed, they are passed through a deep learning model to extract high-dimensional facial representations. Convolutional Neural Networks (CNNs) or pre-trained face recognition models like FaceNet, VGG-Face, or ResNet are commonly used in this step. These models break down the image into feature maps, capturing unique patterns such as facial contours, textures, and landmark positions. This process ensures that every face is encoded into a mathematical representation, allowing for precise comparisons beyond simple pixel matching. The extracted embeddings serve as the foundation for detecting subtle discrepancies between genuine and morphed images.

4.3.4. Feature Combination

After extracting facial features from both the reference and live capture images, the system combines and compares them to detect inconsistencies. Various feature fusion techniques such as concatenation, weighted averaging, or similarity scoring (e.g., Euclidean distance, Cosine Similarity) are applied to evaluate the level of resemblance. If the extracted features align closely, the images are likely genuine; however, any deviations may indicate morphing attempts. This step plays a vital role in distinguishing minor variations caused by natural expressions from deliberate alterations made through face morphing techniques. Feature aggregation methods help improve robustness by integrating multi-layer information from CNN models.

4.3.5. Machine Learning-Based Decision Making

The fused feature vectors are then passed through a classification model trained to distinguish between bona fide and morphed faces. Traditional classifiers like

Support Vector Machines (SVM), Random Forest, or ensemble learning methods can be used for this decision-making process. Additionally, deep neural networks (DNNs) or Transformer-based architectures can enhance accuracy by learning complex relationships between facial features. The classifier is trained on a dataset containing both real and morphed images, allowing it to generalize well and detect sophisticated morphing attacks. The classifier may incorporate probabilistic confidence scores to indicate the likelihood of an image being morphed.

4.3.6. Output Classification: Bona Fide or Morph

Based on the analysis, the system classifies the input as either "Bona Fide" (authentic, accept) or "Morph" (manipulated, reject). If the facial features match within an acceptable threshold, the user is granted authentication. However, if inconsistencies or unnatural blending effects are detected, the system flags the image as a morph and denies access. This final decision ensures that morphed identities are effectively filtered out, strengthening biometric security against identity fraud and unauthorized access. Threshold values for classification can be dynamically adjusted based on real-time scenarios. The system may provide additional feedback explaining rejection reasons to improve user experience. Blockchain technology could be integrated to ensure traceability and immutability of identity verification logs.

4.4 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL / SYSTEM

Software Implementation

The implementation of the deep learning-based face morphing detection system requires a combination of machine learning frameworks, image processing libraries, and backend tools to ensure seamless functionality. The system is designed to handle image acquisition, feature extraction, classification, and decision-making using deep learning models. Below is a detailed description of the software components used in the implementation.

The implementation process is divided into five major phases:

4.4.1 Data Collection and Preprocessing

The first phase involves gathering a diverse dataset containing both genuine and morphed facial images from publicly available repositories and synthetic data generation techniques. Preprocessing includes image normalization, resizing, and alignment to ensure uniformity across samples. Facial landmark detection using Dlib or OpenCV ensures that facial features are properly aligned before further processing. Additionally, image augmentation techniques such as rotation, flipping, and contrast adjustments are applied to enhance the model's robustness. Noise reduction techniques and grayscale conversion are also performed to eliminate redundant information. After preprocessing, the data is split into training, validation, and testing sets to facilitate deep learning model training.

4.4.2. Feature Extraction and Model Selection

In this phase, deep learning-based models such as Convolutional Neural Networks (CNNs), ResNet, VGG-Face, and FaceNet are explored for extracting deep facial features. These models help identify subtle inconsistencies in morphed images that may not be detectable through conventional techniques. Feature extraction is performed at different layers of CNNs to capture both low-level texture variations and high-level structural patterns in facial images.

4.4.3. Model Training and Evaluation

Once the feature extraction process is complete, the system undergoes extensive training using machine learning and deep learning classifiers. Pre-trained models such as VGG16, ResNet50, and EfficientNet are fine-tuned using transfer learning techniques to improve accuracy with minimal training data. The classification phase employs Support Vector Machines (SVM), Random Forest, and XGBoost to distinguish between real and morphed faces. Hyperparameter tuning techniques like grid search and random search are used to optimize model performance.

4.4.4. System Integration and API Development

The trained model is integrated into a real-time face verification system using backend frameworks like Flask, FastAPI, or Django to provide API-based access. This phase involves developing a user-friendly interface where users can upload images for morphing detection. The API endpoints allow secure image processing, feature extraction, and classification while ensuring low-latency responses for real-time applications. The system is deployed on cloud platforms such as AWS, Google Cloud, or Microsoft Azure, enabling scalability and high availability.

4.4.5. Testing, Deployment, and Future Enhancements

The final phase involves rigorous system testing to ensure robustness and efficiency across different operational scenarios. Functional testing verifies that all features, including image preprocessing, feature extraction, classification, and API responses, work as expected. Performance testing assesses response time and system efficiency under high loads. After successful validation, the system is deployed in real-world applications such as airport security, banking verification, and online identity authentication. Future enhancements include integrating blockchain for secure identity verification, improving the model with GAN-based adversarial training, and extending the system for multi-modal biometric verification using voice and fingerprint recognition.

4.5 SOFTWARE COMPONENTS AND TOOLS USED

The software stack consists of essential programming libraries and cloud-based tools to streamline the model development, training, and inference processes.

4.5.1. Programming Languages & Libraries

- Python – Used for implementing deep learning models, preprocessing, and backend API development.
- JavaScript (React.js) – For developing a user-friendly frontend interface.
- SQL/NoSQL (PostgreSQL, MongoDB) – Database for storing user data and verification logs.
- PyTorch – Alternative deep learning framework for experimentation with advanced models.

- Histogram of Oriented Gradients (HOG) – For detecting structural inconsistencies in morphed faces.
- OpenCV – For image processing, face alignment, and feature extraction.
- Dlib – For facial landmark detection and preprocessing.
- Scikit-learn – For implementing classification algorithms such as SVM, Random Forest, and XGBoost.
- Flask / FastAPI / Django – For developing APIs to handle image uploads and processing.
- AWS (EC2, Lambda, S3) – For cloud hosting and model deployment.
- Google Cloud AI Platform – For training deep learning models on GPUs.

CHAPTER 5

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

5.1 SYSTEM STUDY AND TESTING

The system study for deep learning-based face morphing detection involves analyzing existing identity verification methods and their vulnerabilities to morphing attacks. Traditional biometric systems often rely on simple feature matching techniques, which are insufficient to detect morphed faces. This study highlights the need for advanced deep learning models like CNNs to analyze pixel-level inconsistencies and structural distortions in facial images. The system architecture is designed to ensure efficient data preprocessing, feature extraction, and classification, making it robust against sophisticated attacks.

Testing plays a crucial role in evaluating the performance and reliability of the detection system. The model undergoes extensive training and validation using large datasets containing both real and morphed images. Testing phases include unit testing to verify individual components, integration testing to ensure seamless interaction between the frontend, backend, and database, and performance testing to assess accuracy, speed, and scalability. Various evaluation metrics such as precision, recall, F1-score, and ROC curves are used to measure detection efficiency. Adversarial testing is also conducted to analyze how well the system can adapt to new morphing techniques and potential threats. Finally, user acceptance testing (UAT) ensures that the system meets security requirements and provides accurate real-time results for identity verification applications. The system study for deep learning-based face morphing detection involves analyzing existing identity verification methods and their vulnerabilities to morphing attacks. Traditional biometric systems often rely on simple feature matching techniques, which are insufficient to detect morphed faces. This study highlights the need for advanced deep learning models like CNNs to analyze pixel-level inconsistencies and structural distortions in facial images.

5.2 PERFORMANCE ANALYSIS

The performance analysis of deep learning-based face morphing detection focuses on evaluating its accuracy, efficiency, and robustness against different attack techniques. The system's accuracy is measured using key metrics such as precision, recall, F1-score, and ROC-AUC. The deep learning model, particularly CNNs, has demonstrated high effectiveness in distinguishing between real and morphed images by analyzing pixel-level details. Experimental results indicate a significant reduction in false positives and false negatives compared to traditional biometric verification methods, ensuring more reliable identity verification. The system has been tested across various morphing techniques, proving its adaptability and robustness against evolving threats.

Another critical aspect of performance analysis is computational speed and latency. The system is designed to work in real-time, making it essential to evaluate its inference time under different hardware configurations such as CPUs, GPUs, and cloud-based environments. Performance optimization techniques, including model quantization and pruning, help improve speed without compromising accuracy. Results show that GPU-based implementations significantly enhance processing efficiency, making the system viable for real-time applications. Reducing latency is crucial for security-sensitive applications such as airport security and financial transactions, where quick and accurate identity verification is required.

Robustness against external factors such as lighting conditions, image resolution, facial angles, and occlusions is also examined. The system is trained on diverse datasets to ensure that it generalizes well across different environments. Performance evaluation reveals that higher-resolution images lead to better detection rates, while extreme lighting conditions and occlusions can slightly impact accuracy. To mitigate these issues, data augmentation techniques are employed, improving the model’s ability to detect morphing under varying conditions. The system is further tested with different datasets to assess its effectiveness across a

broad range of demographic variations.

A comparative analysis with existing biometric security methods highlights the superiority of deep learning-based morphing detection. Traditional methods relying on geometric feature matching and handcrafted descriptors struggle with detecting subtle alterations introduced by face morphing. The proposed system demonstrates significantly higher accuracy and adaptability, reducing the false acceptance rate and improving identity verification reliability. Incorporating adversarial training enhances the model's robustness against sophisticated attacks, making it a more secure solution compared to conventional approaches.

Scalability is another essential factor in evaluating performance. The system is designed to be deployed in various environments, including edge devices, cloud-based solutions, and large-scale biometric verification systems. Cloud deployment using platforms like AWS and Google Cloud enables faster processing and storage, while edge AI implementations allow offline detection on low-power devices. The system's efficient resource utilization ensures smooth operation, even when handling large datasets or multiple verification requests simultaneously. Performance optimization techniques such as batch processing and parallel computing further enhance its scalability for high-demand applications. Future improvements can further optimize the system for enhanced detection capabilities. The integration of transformer-based models, such as Vision Transformers (ViTs), can refine feature extraction and increase detection accuracy. Additionally, ensemble learning techniques can be employed to combine multiple models, further improving overall performance. Updating the dataset continuously will help the system adapt to emerging face morphing techniques and new attack strategies.

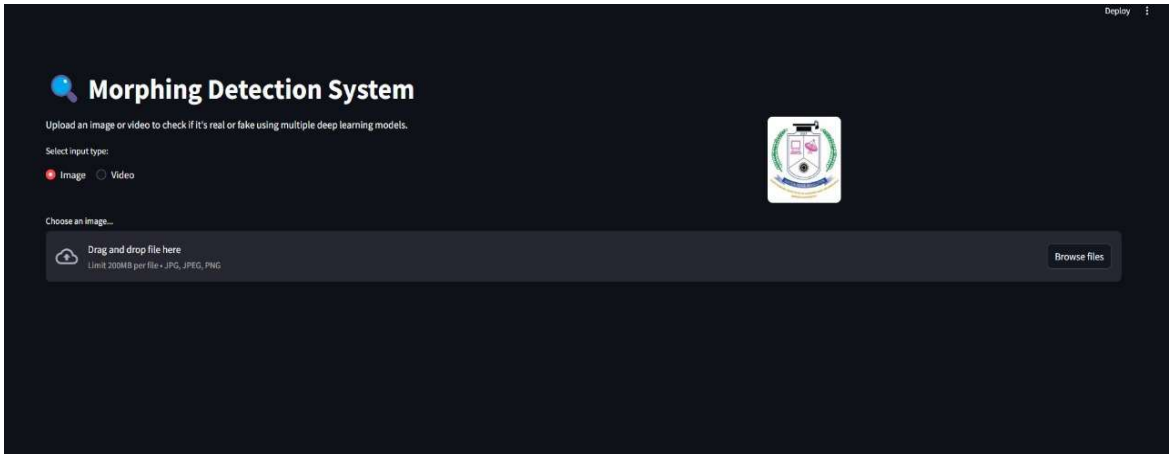

Fig 5.1 User Interface of the Face Morphing Detection

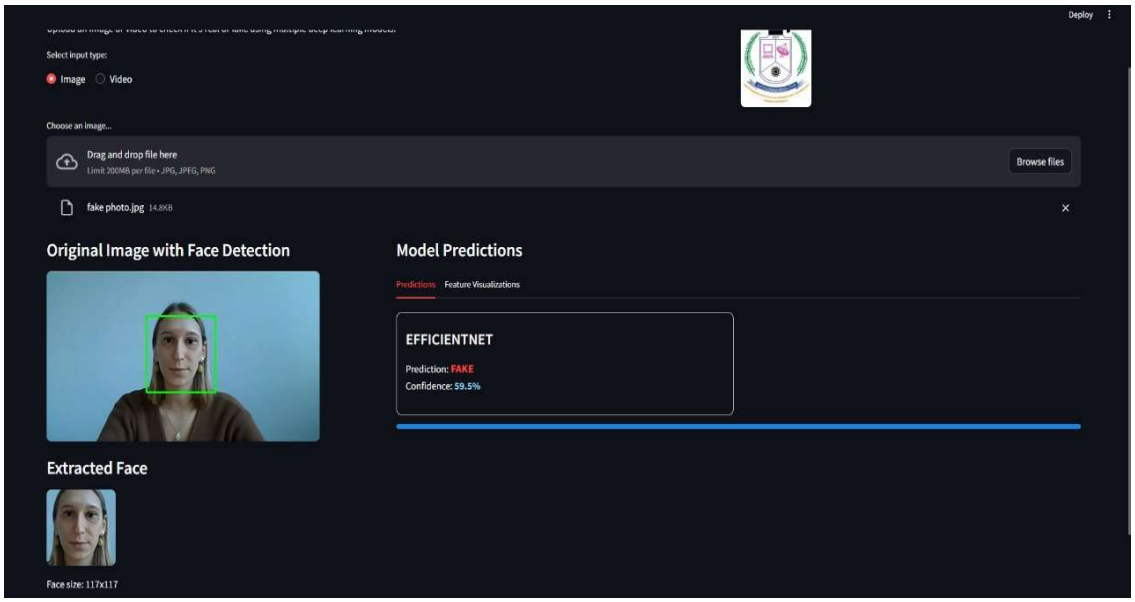

Fig 5.2 Fake Image Detection

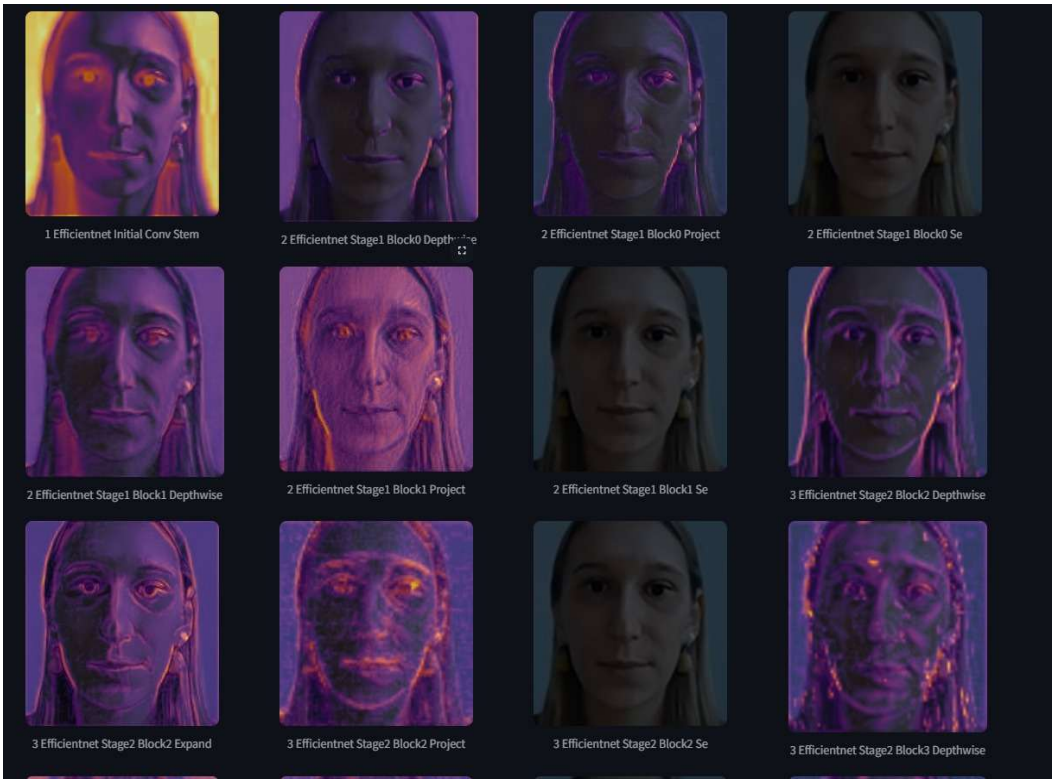

Fig 5.3 Internal Feature Representations of Fake Image

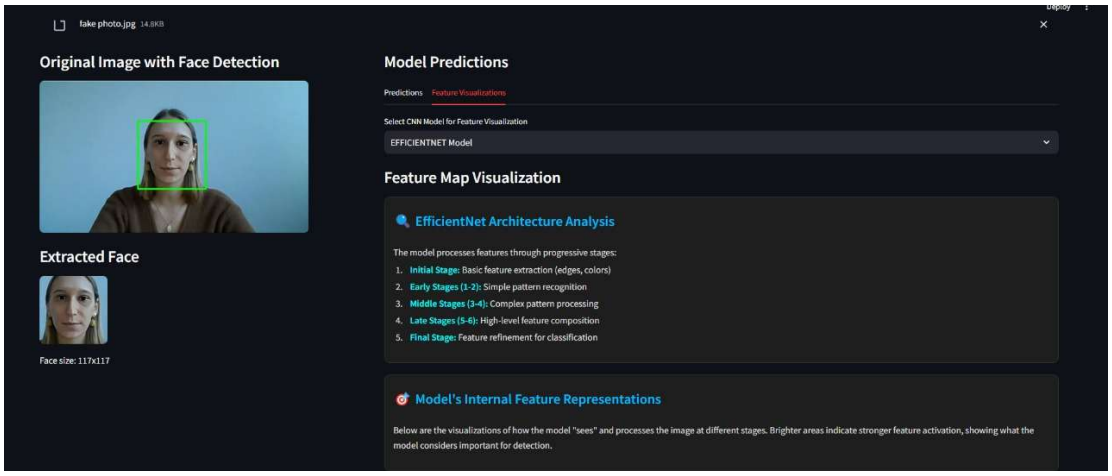

Fig 5.4 EfffficientNet Analysis of Real Image

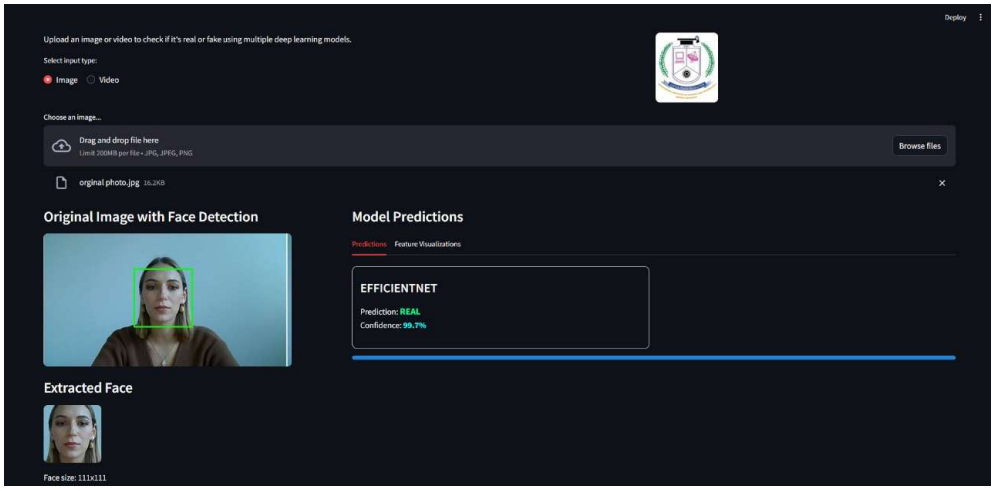

Fig 5.5 Real Image Detection

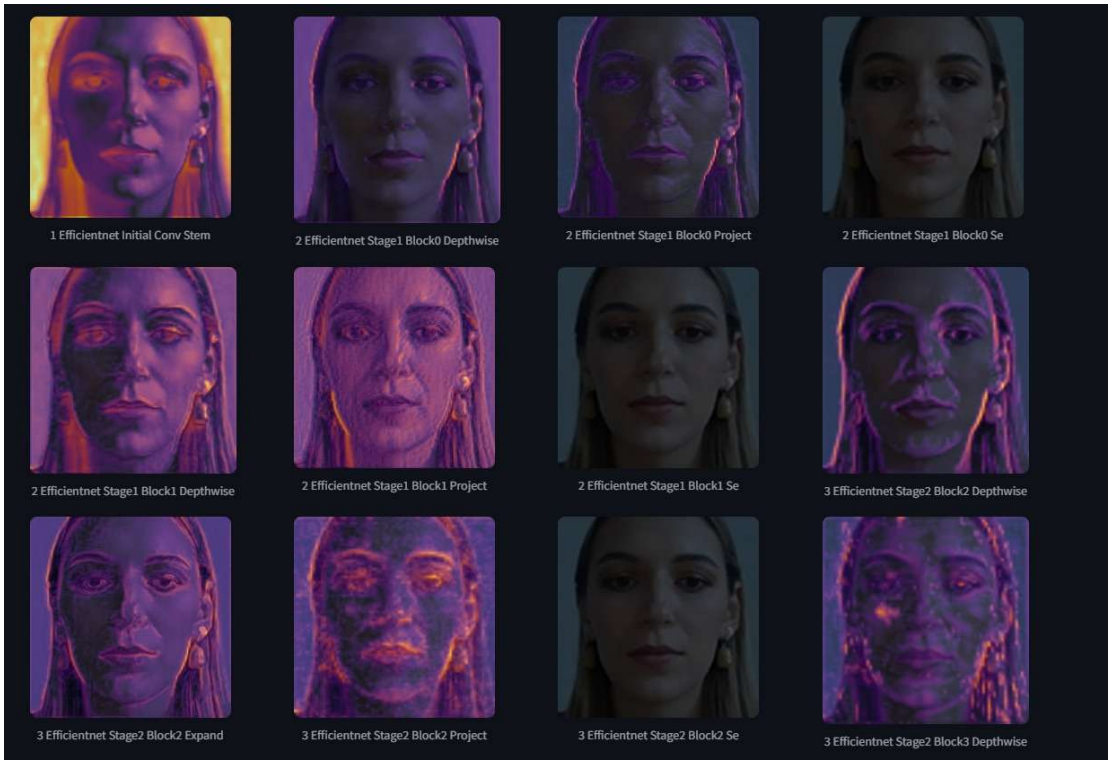

Fig 5.6 Internal Feature Representations of Real Image

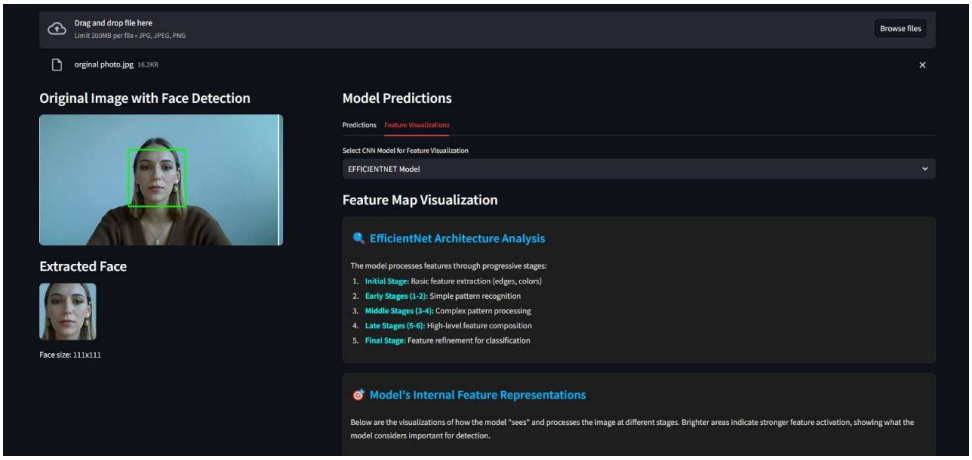

Fig 5.7 EffficientNet Analysis of Real Image

CHAPTER 6

SUMMARY AND CONCLUSIONS

The rapid advancement of deep learning technologies has revolutionized face morphing detection, significantly enhancing the robustness and reliability of identity verification systems. By integrating sophisticated methodologies such as Convolutional Neural Networks (CNNs) and Capsule Networks (Caps Nets), current systems can effectively analyze facial images to identify subtle morphing artifacts. These techniques enable detailed analysis of pixel-level and structural inconsistencies, allowing for the accurate differentiation between genuine and manipulated faces. The application of these advanced models marks a substantial improvement over traditional biometric methods, which often struggle to detect such nuanced alterations. Generative Adversarial Networks (GANs) have played a pivotal role in augmenting the training datasets with realistic morphed images, addressing the challenge of data scarcity. This synthetic data generation enhances the model's ability to recognize a wide range of morphing techniques and improves overall detection accuracy. Additionally, GANs contribute to adversarial training strategies, making models more resilient to sophisticated morphing attacks. As a result, these advancements have significantly strengthened the effectiveness of face morphing detection systems against evolving threats.

6.1 Future Work

Future work in deep learning-based face morphing detection will focus on enhancing model accuracy, generalization, and real-time processing capabilities. One key area of improvement is the integration of transformer-based architectures, such as Vision Transformers (ViTs), which have demonstrated superior feature extraction capabilities in image analysis. These models can complement CNNs and Capsule Networks by capturing long-range dependencies in facial structures, leading to improved detection performance. Additionally, the incorporation of ensemble learning techniques, where multiple models work together, can further enhance robustness against complex morphing techniques.

REFERENCES

1. A. Maithresh, V. Nikhil, H. Saipuneeth and G. V. S. Reddy, "Exploring the Superiority of CapsNet over CNN For Early Detection of Lung Cancer A Comparative Inventive Computation Technologies (ICICT), Lalitpur, Nepal, 2023, pp. 322-329, doi: 10.1109/ICICT57646.2023.10134277.

2. D. Gharde, M. P. A, S. N and S. K. S,"Detection of Morphed Face, Body, Audio signals using Deep Neural Networks," 2022 IEEE 7th International conference for Convergence in Technology (I2CT), Mumbai, India, 2022, pp. 1-6, doi: 10.1109/I2CT54291.2022.9825423.

3. D. K. S and M. T. I, "Deep learning architectures for Brain Tumor detection: A Survey," 2023 Advanced Computing and Communication Technologies for High Performance Applications.

4. F. Mokhayeri, E. Granger and G. -A. Bilodeau, "Synthetic face generation under various operational conditions in video surveillance," 2015 IEEE International Conference on Image Processing (ICIP), Quebec City, QC, Canada, 2015, pp. 4052- 4056, doi: 10.1109/ICIP.2015.7351567.

5. J. He, Z. Xu, Y. Chen, Z. Pan and I. Chih-Lin, "Work in progress paper: Network deployment and operation based on spatial and temporal traffic model," 9th International Conference on Communications and Networking in China, Maoming, China, 2014, pp.144-147 doi:10.1109/CHINACOM.2014.7054275.

6. M. -H. Kim, D. -S. Wang, S. -T. Wang, S. -H. Park and C. -G. Lee, "Improving the Robustness of the Bug Triage Model through Adversarial Training," 2022 International Conference on Information Networking (ICOIN), Jeju-si, Korea, Republic of, 2022, pp.478-481, doi: 10.1109/ICOIN53446.2022.9687279.

7. Q. Zhang and H. He, "Research of an Enhanced YOLOv5-Based Algorithm for Garbage Detection in Service Areas," 2024 4th International Conference on Neural Networks, Information and Communication Engineering (NNICE), Guangzhou, China, 2024, pp.889-894, doi:10.1109/NNICE61279.2024.10499157.

8. S. K. Dubey, S. Gupta, S. K. Pandey andS. Mittal, "Ethical Considerations in Data Mining and Database Research," 2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2024, pp. 1-4, doi10.1109/ICRITO61523.2024.10522200.

9. V. Gunturu, N. Maiti, B. Toure, P. Kunekar, S. B. Banu and D. Sahaya Lenin, "Transfer Learning in Biomedical Image Classification," 2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), Chennai, India, 2024, pp. 1-5, doi: 10.1109/ACCAI61061.2024.10601862.

10. X. Chen, Y. Zou and H. Ke, "TrafficYOLO: YOLO with Multi-Head Attention Mechanism for Traffic Detection Scenarios," 2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT), Nanjing, China, 2024, pp.2276-2279, doi: 10.1109/AINIT61980.2024.10581465.

APPENDIX

A. SOURCE CODE

```
import streamlit as st
import torch
import torch.nn as nn
from PIL import Image
import os
import mlflow
from data_handler import get_transforms
import glob
import logging
import sys
import warnings
import mediapipe as mp
import numpy as np
import cv2
from feature_visualization import get_feature_maps,
display_feature_maps
from video_processor import VideoProcessor
import subprocess
import gc
import time
from contextlib import contextmanager

# Configure memory management
torch.backends.cudnn.benchmark = False
torch.backends.cudnn.deterministic = True

# Run setup script if models don't exist
if not os.path.exists("converted_models"):
    try:
        subprocess.run(['bash', 'setup.sh'], check=True)
    except subprocess.CalledProcessError as e:
        st.error(f"Error running setup script: {str(e)}")
        st.stop()

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Suppress specific warnings
warnings.filterwarnings('ignore', category=UserWarning,
module='timm')
warnings.filterwarnings('ignore', category=UserWarning,
module='huggingface_hub')
```

```
warnings.filterwarnings('ignore', category=FutureWarning,
module='torch.serialization')

# Initialize MediaPipe Face Detection
mp_face_detection = mp.solutions.face_detection
mp_drawing = mp.solutions.drawing_utils

# Set page config
st.set_page_config(
    page_title="Morphing Detection",
    page_icon="🔍",
    layout="wide"
)

# Custom CSS for better styling
st.markdown("""
    <style>


        /* Control maximum size of all images */
        .stImage > img {
            max-height: 300px !important;
            max-width: 100% !important;
            width: auto !important;
            object-fit: contain;
        }

        /* Specific control for face grid images */
        .face-grid-image > img {
            max-height: 200px !important;
            max-width: 100% !important;
            width: auto !important;
            object-fit: contain;
        }

        /* Feature map images */
        .feature-map-image > img {
            max-height: 150px !important;
            max-width: 100% !important;
            width: auto !important;
            object-fit: contain;
        }

        div.stMarkdown {
            max-width: 100%;
        }
        div[data-testid="column"] {
            background-color: #1e1e1e;
```

```
            padding: 15px;
            border-radius: 10px;
            margin: 5px;
            border: 1px solid #333;
        }
        div[data-testid="column"] h4 {
            color: #ffffff;
        }
        div[data-testid="column"] p {
            color: #e0e0e0;
        }
    </style>

""", unsafe_allow_html=True)

# Constants
MODEL_IMAGE_SIZES = {
    "efficientnet": 300,
}

SESSION_TIMEOUT = 3600  # 1 hour in seconds
LAST_ACTIVITY_KEY = "last_activity"
SESSION_DATA_KEY = "session_data"

@contextmanager
def cleanup_on_exit():
    """Context manager to ensure cleanup when processing is
done"""
    try:
        yield
    finally:
        # Clear memory
        if torch.cuda.is_available():
            torch.cuda.empty_cache()
        gc.collect()

def init_session_state():
    """Initialize or update session state"""
    if LAST_ACTIVITY_KEY not in st.session_state:
        st.session_state[LAST_ACTIVITY_KEY] = time.time()
    if SESSION_DATA_KEY not in st.session_state:
        st.session_state[SESSION_DATA_KEY] = {}

    # Update last activity
    st.session_state[LAST_ACTIVITY_KEY] = time.time()

def check_session_timeout():
    """Check if session has timed out"""
```

```
    if LAST_ACTIVITY_KEY in st.session_state:
        last_activity = st.session_state[LAST_ACTIVITY_KEY]
        if time.time() - last_activity > SESSION_TIMEOUT:
            # Clear session state
            st.session_state[SESSION_DATA_KEY] = {}
            st.session_state[LAST_ACTIVITY_KEY] = time.time()
            return True
    return False

def clear_session_data():
    """Clear session-specific data"""
    if SESSION_DATA_KEY in st.session_state:
        st.session_state[SESSION_DATA_KEY] = {}
    if torch.cuda.is_available():
        torch.cuda.empty_cache()
    gc.collect()

@st.cache_resource
def get_cached_model(model_path, model_type):
    """Cache and share model instances across sessions"""
    try:
        # Initialize model based on type
        model = None
        if model_type == "efficientnet":
            from train_efficientnet import DeepfakeEfficientNet
            model = DeepfakeEfficientNet()

        if model is None:
            return None

        # Load state dict
        state_dict = torch.load(model_path, map_location='cpu',
weights_only=True)
        model.load_state_dict(state_dict, strict=False)
        model.eval()

        return model
    except Exception as e:
        return None

def extract_face(image, padding=0.1):
    """Extract face from image using MediaPipe with padding"""
    # Convert PIL Image to cv2 format
    img_cv = cv2.cvtColor(np.array(image),
cv2.COLOR_RGB2BGR)
    height, width = img_cv.shape[:2]

    with mp_face_detection.FaceDetection(
```

```
        min_detection_confidence=0.5,
        model_selection=1  # Use full range model for better
detection
    ) as face_detection:
        # Convert the BGR image to RGB
        results = face_detection.process(cv2.cvtColor(img_cv,
cv2.COLOR_BGR2RGB))

        if not results.detections:
            # Try with a lower confidence threshold for video frames
            with mp_face_detection.FaceDetection(
                min_detection_confidence=0.3,
                model_selection=1
            ) as face_detection_lower:
                results =
face_detection_lower.process(cv2.cvtColor(img_cv,
cv2.COLOR_BGR2RGB))
                if not results.detections:
                    return None, None

        # Get the first face detection
        detection = results.detections[0]
        bbox = detection.location_data.relative_bounding_box

        # Calculate padding with bounds checking
        pad_w = max(int(bbox.width * width * padding), 0)
        pad_h = max(int(bbox.height * height * padding), 0)

        # Convert relative coordinates to absolute with padding
        x = max(0, int(bbox.xmin * width) - pad_w)
        y = max(0, int(bbox.ymin * height) - pad_h)
        w = min(int(bbox.width * width) + (2 * pad_w), width - x)
        h = min(int(bbox.height * height) + (2 * pad_h), height - y)

        # Additional checks for valid dimensions
        if w <= 0 or h <= 0 or x >= width or y >= height:
            return None, None

        # Extract face region with padding
        try:
            face_region = img_cv[y:y+h, x:x+w]
            if face_region.size == 0:  # Check if region is empty
                return None, None

            # Convert back to RGB for PIL
            face_region_rgb = cv2.cvtColor(face_region,
cv2.COLOR_BGR2RGB)
            face_pil = Image.fromarray(face_region_rgb)
```

```
            # Create visualization
            img_cv_viz = img_cv.copy()
            cv2.rectangle(img_cv_viz, (x, y), (x+w, y+h), (0, 255, 0), 2)
            img_viz = cv2.cvtColor(img_cv_viz,
cv2.COLOR_BGR2RGB)

            return face_pil, Image.fromarray(img_viz)
        except Exception as e:
            logger.error(f"Error in face extraction: {str(e)}")
            return None, None

def resize_image_for_display(image, max_size=300):
    width, height = image.size
    if width > height:
        if width > max_size:
            ratio = max_size / width
            new_size = (max_size, int(height * ratio))
    else:
        if height > max_size:
            ratio = max_size / height
            new_size = (int(width * ratio), max_size)

    if width > max_size or height > max_size:
        image = image.resize(new_size,
Image.Resampling.LANCZOS)
    return image

def process_image(image, model_type):
    try:
        img_size = MODEL_IMAGE_SIZES.get(model_type, 224)
        transform = get_transforms(img_size)
        transformed_image = transform(image).unsqueeze(0)
        logger.info(f"Successfully processed image for {model_type}
(size: {img_size})")
        return transformed_image
    except Exception as e:
        logger.error(f"Error processing image: {str(e)}")
        return None

def load_model(model_path, model_type):
    """Load a model using the cached function"""
    return get_cached_model(model_path, model_type)

def format_confidence(confidence):
    """Format confidence score with color based on value"""
    if confidence >= 0.8:
        color = "#00ffff"
```

```
    elif confidence >= 0.6:
        color = "#00bfff"
    else:
        color = "#87ceeb"
    return f'<span style="color: {color}; font-weight:
bold;">{confidence:.1%}</span>'

def format_prediction(prediction):
    """Format prediction with color (red for FAKE, green for
REAL)"""
    color = "#ff4444" if prediction == "FAKE" else "#00ff9d"
    return f'<span style="color: {color}; font-weight: bold; font-size:
1.1em;">{prediction}</span>'

def main():

    init_session_state()

    # Check for timeout
    if check_session_timeout():
        st.warning("Your session has timed out. Please reload the
page.")
        return

    st.title("🔍 Morphing Detection System")
    col1, col2 = st.columns([2, 1])  # Left (text) | Right (image)

    with col1:
        st.write("Upload an image or video to check if it's real or fake
using multiple deep learning models.")
        input_type = st.radio("Select input type:", ["Image", "Video"],
horizontal=True)

    if input_type == "Image":
        uploaded_file = st.file_uploader("Choose an image...",
type=["jpg", "jpeg", "png"])

        if uploaded_file is not None:
            with cleanup_on_exit():
                process_image_input(uploaded_file)
    else:
        uploaded_file = st.file_uploader("Choose a video...",
type=["mp4", "avi", "mov"])

        if uploaded_file is not None:
            with cleanup_on_exit():
                process_video_input(uploaded_file)
```

```
    with col2:
        st.image("images/logo.jpg", width=120)



def process_video_input(video_file):
    try:
        # Store video path in session state
        if 'video_path' not in
st.session_state[SESSION_DATA_KEY]:
            st.session_state[SESSION_DATA_KEY]['video_path'] =
None

        # Get number of frames from user with a start button
        col1, col2 = st.columns([3, 1])
        with col1:
            num_frames = st.slider("Number of frames to analyze",
min_value=10, max_value=300, value=100, step=10,
                                   help="More frames = more accurate but
slower processing")
        with col2:
            start_button = st.button("Start Processing",
type="primary")

        if not start_button:
            return

        # Create progress bars
        st.write("### Processing Video")
        progress_load = st.progress(0)
        status_load = st.empty()
        progress_extract = st.progress(0)
        status_extract = st.empty()
        progress_faces = st.progress(0)
        status_faces = st.empty()
        progress_process = st.progress(0)
        status_process = st.empty()

        # Initialize video processor
        video_processor =
VideoProcessor(num_frames=num_frames)

        # Save uploaded video
        video_path =
video_processor.save_uploaded_video(video_file)

        # Load models with progress tracking
        status_load.text("Loading models: Initializing...")
```

```
        progress_load.progress(0.1)

        # Load models
        converted_dir = "converted_models"
        if not os.path.exists(converted_dir):
            st.error("Please run convert_models.py first to convert the
models!")
            return

        model_files = glob.glob(os.path.join(converted_dir,
"*_converted.pth"))
        if not model_files:
            st.error("No converted models found! Please run
convert_models.py first.")
            return

        # Load all models with progress tracking
        models_data = []
        total_models = len(model_files)
        for i, model_path in enumerate(model_files):
            model_type =
os.path.basename(model_path).replace("_converted.pth", "")
            status_load.text(f"Loading models: {model_type.upper()}")
            progress_load.progress((i + 1) / (total_models + 1))

            model = load_model(model_path, model_type)
            if model is not None:
                models_data.append({
                    'model': model,
                    'model_type': model_type,
                    'image_size': MODEL_IMAGE_SIZES[model_type]
                })

        progress_load.progress(1.0)
        status_load.text("Models loaded successfully!")

        if not models_data:
            st.error("No models could be loaded! Please check the
model files.")
            return

        def update_progress(progress_bar, status_placeholder,
stage):
            def callback(progress):
                progress_bar.progress(progress)
                status_placeholder.text(f"{stage}: {progress:.1%}")
            return callback
```

```
        # Process video with progress tracking
        progress_callbacks = {
            'extract_frames': update_progress(progress_extract,
status_extract, "Extracting frames"),
            'extract_faces': update_progress(progress_faces,
status_faces, "Detecting faces"),
            'process_frames': update_progress(progress_process,
status_process, "Processing frames")
        }

        results, frame_results, faces =
video_processor.process_video(
            video_path,
            extract_face_fn=extract_face,
            process_image_fn=process_image,
            models=models_data,
            progress_callbacks=progress_callbacks
        )

        # Clear progress bars and status messages
        progress_load.empty()
        status_load.empty()
        progress_extract.empty()
        status_extract.empty()
        progress_faces.empty()
        status_faces.empty()
        progress_process.empty()
        status_process.empty()

        if results:
            # Create tabs for different views
            pred_tab, faces_tab = st.tabs(["Predictions", "Detected
Faces"])

            with pred_tab:
                st.write("### Model Predictions")
                cols = st.columns(2)
                col_idx = 0

                for result in results:
                    with cols[col_idx % 2]:
                        st.markdown(f"""
                        <div style='padding: 15px; border-radius: 10px;
border: 1px solid #ddd; margin: 5px 0;'>
                            <h4>{result['model_type'].upper()}</h4>
                            <p>Overall Prediction:
{format_prediction(result['prediction'])}<br>
                            Average Confidence:
```

```
{format_confidence(result['confidence'])}<br>
                        Fake Frames:
{result['fake_frame_ratio']:.1%}</p>
                    </div>
                    """, unsafe_allow_html=True)
                col_idx += 1

        with faces_tab:
            st.write("### Sample Detected Faces")
            # Display a subset of detected faces in a grid
            n_sample_faces = min(12, len(faces))  # Show up to 12
faces
            sample_indices = np.linspace(0, len(faces)-1,
n_sample_faces, dtype=int)

            # Create grid layout for faces in video processing
            cols = st.columns(4)  # 4 faces per row
            for idx, face_idx in enumerate(sample_indices):
                with cols[idx % 4]:
                    face = faces[face_idx]
                    resized_face = resize_image_for_display(face,
max_size=200)
                    st.markdown('<div class="face-grid-image">',
unsafe_allow_html=True)
                    st.image(resized_face, caption=f"Frame
{face_idx}", use_container_width=False)
                    st.markdown('</div>', unsafe_allow_html=True)
        else:
            st.warning("No faces could be detected in the video
frames.")

        # Cleanup
        if st.session_state[SESSION_DATA_KEY].get('video_path'):
            try:

os.unlink(st.session_state[SESSION_DATA_KEY]['video_path'])
                st.session_state[SESSION_DATA_KEY]['video_path'] =
None
            except:
                pass

        clear_session_data()

    except Exception as e:
        st.error(f"An error occurred: {str(e)}")
        logger.error(f"Error processing video: {str(e)}")
        clear_session_data()
```

```
def process_image_input(uploaded_file):
    try:
        with cleanup_on_exit():
            # Load and display the image
            image = Image.open(uploaded_file).convert('RGB')

            # Extract face
            face_image, viz_image = extract_face(image)

            if face_image is None:
                st.error("No face detected in the image. Please upload
an image containing a clear face.")
                return

            # Create two columns for image and info
            col1, col2 = st.columns([1, 2])

            with col1:
                # Display original image with face detection
                st.write("### Original Image with Face Detection")
                st.image(viz_image, use_container_width=False)

                # Display extracted face
                st.write("### Extracted Face")
                display_face = resize_image_for_display(face_image)
                st.image(display_face, use_container_width=False)
                st.write(f"Face size:
{face_image.size[0]}x{face_image.size[1]}")

            with col2:
                # Load all converted models
                converted_dir = "converted_models"
                if not os.path.exists(converted_dir):
                    st.error("Please run convert_models.py first to
convert the models!")
                    return

                # Find all converted model files
                model_files = glob.glob(os.path.join(converted_dir,
"*_converted.pth"))
                if not model_files:
                    st.error("No converted models found! Please run
convert_models.py first.")
                    return

                st.write("### Model Predictions")

                # Create tabs for predictions and visualizations
```

```
            pred_tab, viz_tab = st.tabs(["Predictions", "Feature
Visualizations"])

            with pred_tab:
                # Create columns for results
                cols = st.columns(2)
                col_idx = 0
                models_loaded = False

                progress_bar = st.progress(0)
                predictions_data = []

                for i, model_path in enumerate(model_files):
                    # Get model type from filename
                    model_type =
os.path.basename(model_path).replace("_converted.pth", "")

                    with st.spinner(f'Processing with
{model_type.upper()}...'):
                        model = load_model(model_path, model_type)
                        if model is None:
                            continue

                        models_loaded = True


                        processed_image =
process_image(face_image, model_type)
                        if processed_image is None:
                            st.error(f"Failed to process image for
{model_type}")
                            continue


                        try:
                            with torch.no_grad():
                                output = model(processed_image)
                                probability = torch.sigmoid(output).item()
                                prediction = "FAKE" if probability > 0.5 else
"REAL"
                                confidence = probability if prediction ==
"FAKE" else 1 - probability

                            # Store prediction data
                            predictions_data.append({
                                'model_type': model_type,
                                'prediction': prediction,
                                'confidence': confidence,
```

```
                        'model': model,
                        'processed_image': processed_image
                    })

                    # Display results in card format
                    with cols[col_idx % 2]:
                        st.markdown(f"""
                        <div style='padding: 15px; border-radius:
10px; border: 1px solid #ddd; margin: 5px 0;'>
                            <h4>{model_type.upper()}</h4>
                            <p>Prediction:
{format_prediction(prediction)}<br>
                            Confidence:
{format_confidence(confidence)}</p>
                        </div>
                        """, unsafe_allow_html=True)
                    col_idx += 1

                except Exception as e:
                    st.error(f"Error making prediction with
{model_type}: {str(e)}")

                progress_bar.progress((i + 1) / len(model_files))

        with viz_tab:
            if predictions_data:
                # Filter for CNN models only
                cnn_predictions = [p for p in predictions_data
                                   if p['model_type'].lower() in
['efficientnet']]

                if cnn_predictions:
                    # Add model selector for CNN models only
                    selected_model = st.selectbox(
                        "Select CNN Model for Feature Visualization",
                        options=[p['model_type'].upper() for p in
cnn_predictions],
                        format_func=lambda x: f"{x} Model"
                    )

                    # Get selected model data
                    model_data = next(p for p in cnn_predictions if
p['model_type'].upper() == selected_model)

                    st.write("### Feature Map Visualization")

                    # Add model-specific descriptions
                    if model_data['model_type'].lower() ==
```

```
'efficientnet':
                    st.markdown("""
                    <div style='background-color: #1e1e1e;
padding: 15px; border-radius: 10px; border: 1px solid #333;'>
                        <h4 style='color: #00bfff; margin-bottom:
10px;'>🔍 EfficientNet Architecture Analysis</h4>
                        <p style='color: #e0e0e0; margin-bottom:
5px;'>The model processes features through progressive
stages:</p>
                        <ol style='color: #e0e0e0;'>
                            <li><strong style='color: #00ffff;'>Initial
Stage:</strong> Basic feature extraction (edges, colors)</li>
                            <li><strong style='color: #00ffff;'>Early
Stages (1-2):</strong> Simple pattern recognition</li>
                            <li><strong style='color: #00ffff;'>Middle
Stages (3-4):</strong> Complex pattern processing</li>
                            <li><strong style='color: #00ffff;'>Late
Stages (5-6):</strong> High-level feature composition</li>
                            <li><strong style='color: #00ffff;'>Final
Stage:</strong> Feature refinement for classification</li>
                        </ol>
                    </div>
                    """, unsafe_allow_html=True)


                st.markdown("""
                <div style='background-color: #1e1e1e;
padding: 15px; border-radius: 10px; border: 1px solid #333;
margin-top: 20px;'>
                    <h4 style='color: #00bfff; margin-bottom:
10px;'>🎯 Model's Internal Feature Representations</h4>
                    <p style='color: #e0e0e0;'>
                        Below are the visualizations of how the
model "sees" and processes the image at different stages.
                        Brighter areas indicate stronger feature
activation, showing what the model considers important for
detection.
                    </p>
                </div>
                """, unsafe_allow_html=True)

                # Get feature maps
                visualizations = get_feature_maps(
                    model_data['model'],
                    model_data['model_type'],
                    model_data['processed_image']
                )
```

```
                    if visualizations:
                        # Sort visualizations by architectural order
                        if model_data['model_type'].lower() ==
'xception':
                            # Sort by flow order
                            sorted_maps = {}
                            for k, v in visualizations.items():
                                if 'entry_' in k:
                                    sorted_maps[k.replace('entry_', '1_')]
= v
                                elif 'middle_' in k:
                                    sorted_maps[k.replace('middle_', '2_')]
= v
                                elif 'exit_' in k:
                                    sorted_maps[k.replace('exit_', '3_')] =
v

                            # Display all maps in order
                            display_feature_maps(face_image,
dict(sorted(sorted_maps.items())))

                        else:  # EfficientNet
                            # Sort by stage order
                            sorted_maps = {}
                            stage_order = {
                                'initial': '1',
                                'stage1': '2',
                                'stage2': '3',
                                'stage3': '4',
                                'stage4': '5',
                                'stage5': '6',
                                'stage6': '7',
                                'final': '8',
                                'conv_head': '9'
                            }

                            for k, v in visualizations.items():
                                for stage, order in stage_order.items():
                                    if stage in k:
                                        sorted_maps[f"{order}_{k}"] = v
                                        break

                            # Display all maps in order
                            display_feature_maps(face_image,
dict(sorted(sorted_maps.items())))
                    else:
                        st.info("No feature maps available for this
model.")
```

```
                    else:
                        st.info("Please select a CNN model
    (EfficientNet) for feature visualization.")
                else:
                    st.warning("No models available for visualization.")

                if not models_loaded:
                    st.error("No models could be loaded! Please check
    the model files.")

        except Exception as e:
            st.error(f"An error occurred: {str(e)}")
            logger.error(f"Error in process_image_input: {str(e)}")
            clear_session_data()

    if __name__ == "__main__":
main()
```