INDUSTRIAL ORIENTED MINI PROJECT

Report

On

MULTI-MODEL DEEPFAKE DETECTION USING MTCNN

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

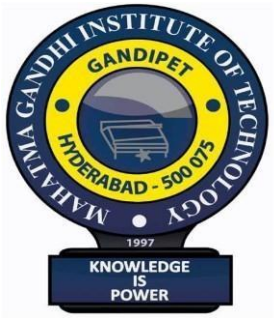
Anthamgari Tejasri - 22261A1204

Challuri Bhanuprakash - 23265A1202

Under the guidance of

Mrs. J. Hima Bindu

Assistant Professor, Department of IT



DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **MULTI-MODEL DEEPFAKE DETECTION USING MTCNN** submitted by **Anthamgari Tejasri (22261A1204), Challuri Bhanuprakash (23265A1202)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Mrs. J. Hima Bindu**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:

Mrs. J. Hima Bindu
Assistant Professor
Dept. of IT

IOMP Supervisor:

Dr. U. Chaitanya
Assistant Professor
Dept. of IT

EXTERNAL EXAMINAR

Dr. D. Vijaya Lakshmi
Professor and HOD
Dept. of IT

DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **MULTI-MODEL DEEPFAKE DETECTION USING MTCNN** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Mrs. J. Hima Bindu,** Assistant Professor, Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

Anthamgari Tejasri - 22261A1204

Challuri Bhanuprakash – 23265A1202

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our internal supervisor **Mrs. J. Hima Bindu,** Assistant Professor, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our Project Coordinator(s) **Dr. U. Chaitanya**, Assistant Professor, Department of IT, for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

Anthamgari Tejasri - 22261A1204

Challuri Bhanuprakash -23265A1202

ABSTRACT

With the rise of advanced computer vision and natural language processing techniques, deep learning (DL) models have enabled the creation of highly realistic deepfake content, including manipulated images, videos, and audio. These deepfakes are increasingly used to spread misinformation, propaganda, and disinformation, posing significant threats to personal and organizational reputations. Existing research has predominantly focused on detecting deepfake images and videos, often overlooking the importance of audio-visual synchronization. However, MTCNN (Multi-Task cascaded Convolutional Networks) are now capable of producing highly convincing audio-visual deepfakes, necessitating robust multi-modal detection approaches.

This project presents a comprehensive study of multi-modal deepfake detection using a combination of audio and visual cues to improve detection accuracy. We explore state-of-the-art machine learning (ML) and deep learning models that analyse both speech patterns and facial movements to identify inconsistencies. By leveraging multi-modal fusion techniques, our approach enhances the detection of subtle manipulations that are challenging to detect using single-modality methods. Additionally, we provide a comparative analysis of public datasets suitable for multi-modal deepfake detection, including those containing synchronized audio and video manipulations.

We also discuss implementation challenges such as data imbalance, computational complexity, and the need for real-time detection capabilities. To address these challenges, we explore optimized architectures, including attention mechanisms and transformer-based models, for efficient feature extraction and fusion. A unique case study, IBMM, is presented to demonstrate the effectiveness of our proposed approach.

LIST OF FIGURES

Fig. 3.2.1	Architecture of deepfake detection using MTCNN	10
Fig. 3.3.1.1	Use Case Diagram	12
Fig. 3.3.2.1	Class Diagram	13
Fig. 3.3.3.1	Activity Diagram	15
Fig. 3.3.4.1	Sequence Diagram	17
Fig. 3.3.5.1	Component Diagram	19
Fig. 3.3.6.1	Deployment Diagram	21
Fig. 4.2.1	Installation of the required dependencies	30

LIST OF TABLES

Table 2.1	Literature Survey of Research papers	7
Table 5.1	Test Cases	34

TABLE OF CONTENTS

Chapter No	Title	Page No
	CERTIFICATE	**i**
	DECLARATION	**ii**
	ACKNOWLEDGEMENT	**iii**
	ABSTRACT	**iv**
	LIST OF FIGURES	**v**
	LIST OF TABLES	**vi**
1	**INTRODUCTION**	**1**
	1.1 MOTIVATION	1
	1.2 PROBLEM STATEMENT	1
	1.3 EXISTING SYSTEM	2
	1.3.1 LIMITATIONS	2
	1.4 PROPOSED SYSTEM	3
	1.4.1 ADVANTAGES	3
	1.5 OBJECTIVES	4
	1.6 HARDWARE AND SOFTWARE REQUIREMENTS	4
2	**LITERATURE SURVEY**	**6**
3	**ANALYSIS AND DESIGN**	**8**
	3.1 MODULES	8
	3.2 ARCHITECTURE	10
	3.3 UML DIAGRAMS	11
	3.3.1 USE CASE DIAGRAM	11
	3.3.2 CLASS DIAGRAM	13
	3.3.3 ACTIVITY DIAGRAM	14
	3.3.4 SEQUENCE DIAGRAM	17
	3.3.5 COMPONENT DIAGRAM	19

Chapter No	Title	Page No
	3.3.6 DEPLOYMENT DIAGRAM	20
	3.4 METHODOLOGY	23
4	**CODE AND IMPLEMENTATION**	**25**
	4.1 CODE	25
	4.2 IMPLEMENTATION	28
5	**TESTING**	**31**
	5.1 INTRODUCTION TO TESTING	31
	5.2 TEST CASES	34
6	**RESULTS**	35
7	**CONCLUSION AND FUTURE ENHANCEMENTS**	38
	7.1 CONCLUSION	**38**
	7.2 FUTURE ENHANCEMENTS	38
8	**REFERENCES**	40

1.INTRODUCTION

1.1 MOTIVATION

In recent years, the proliferation of deepfake technologies has raised serious concerns across various domains including media, politics, cybersecurity, and personal privacy. Deepfakes—synthetically generated or altered content using deep learning—are now capable of producing highly realistic audio and visual manipulations. These fake media assets can be used to impersonate individuals, spread misinformation, and manipulate public opinion. While existing detection methods have focused primarily on analysing visual content, attackers are increasingly leveraging sophisticated models like MTCNN to create audio-visual deepfakes that maintain strong lip-sync consistency and natural speech patterns, making detection even more difficult.

The need for robust detection mechanisms that consider both audio and visual streams has never been more critical. As single-modality approaches often fail to capture subtle cross-modal inconsistencies, a multi-modal deepfake detection framework is essential to enhance detection accuracy and reliability. This project is driven by the urgent demand for effective countermeasures that can analyse and detect such highly convincing manipulations in real-time applications.

1.2 PROBLEM STATEMENT

With the increasing sophistication of deepfake generation techniques, malicious actors can now create highly realistic and synchronized audio-visual content that is nearly indistinguishable from authentic media. While most existing deepfake detection methods focus primarily on either image or video analysis, they often fail to detect manipulations that maintain strong audio-visual consistency. Current approaches lack the ability to fully capture and analyse the complex interplay between facial expressions and speech, making them ineffective against multi-modal deepfakes generated using advanced models like MTCNN.

This research addresses the critical gap in deepfake detection by developing a robust multi-modal detection framework that simultaneously analyses both audio and visual cues. The challenge lies in effectively fusing these heterogeneous data streams to detect subtle inconsistencies, while also addressing issues such as data imbalance, high computational requirements, and the need for real-time performance. This project aims to build an efficient and accurate detection system leveraging deep learning architectures, including attention-based

and transformer models, to improve reliability and resilience against increasingly deceptive deepfakes.

1.3 EXISTING SYSTEM

Deepfake detection systems have traditionally focused on either visual or audio cues independently. Visual-based systems, such as those built using FaceForensics++, XceptionNet, MesoNet, and EfficientNet, analyze facial features, expressions, and frame-level artifacts to identify manipulations in images and videos. These models can detect inconsistencies like unnatural blinking, texture irregularities, or lighting mismatches.

Separately, audio-based detection systems, like those developed for the ASVspoof challenge, focus on speech signal analysis using features such as Mel-Frequency Cepstral Coefficients (MFCCs) and spectrograms to detect synthesized or converted speech. These approaches rely on machine learning models such as CNNs or LSTMs to classify the authenticity of audio samples. Recently, some research has explored multi-modal deepfake detection using datasets like FakeAVCeleb and DeepFakeTIMIT, which incorporate both audio and video. These systems generally use separate models to extract features from each modality and then apply basic fusion methods such as feature concatenation for final classification..

1.3.1 Limitations

- **Single-Modality Dependency**
  Most systems focus on either audio or video, failing to detect inconsistencies between lip movements and speech.
- **Weak Multi-Modal Fusion Techniques**
  Fusion methods are often shallow (e.g., concatenation), unable to effectively capture complex relationships between modalities.
- **High Computational Requirements**
  Deep learning models used for high-resolution video and audio analysis require significant processing power, limiting real-time deployment.
- **Lack of High-Quality Multi-Modal Datasets**
  There are very few comprehensive datasets with synchronized audio and video deepfakes, hindering model training and evaluation.
- **Overfitting and Poor Generalization**
  Models trained on limited datasets tend to overfit and fail to detect deepfakes in real-world or previously unseen data.

1.4 PROPOSED SYSTEM

To effectively detect sophisticated audio-visual deepfakes generated using MTCNN, we propose a robust multi-modal deepfake detection system that integrates both audio and visual information for improved accuracy. The system begins by accepting synchronized audio and video input. From the video stream, facial regions are accurately detected and extracted using MTCNN, ensuring consistent and high-quality face tracking across frames. Simultaneously, the corresponding audio is preprocessed and temporally aligned with the video to maintain synchronization.

Feature extraction is performed separately for both modalities. For the visual stream, deep convolutional networks such as InceptionResNetV1 or 3D CNNs are used to capture spatial and temporal features from face sequences, focusing on lip movement, facial expressions, and subtle anomalies. On the audio side, features like Mel-spectrograms or MFCCs are extracted and analysed using recurrent networks like LSTM or transformer-based models to detect anomalies in speech patterns and voice tone.

Finally, the fused features are passed through a fully connected neural network with a softmax or sigmoid output to classify the content as real or fake. Optionally, explainability can be incorporated using techniques like Grad-CAM to visualize the regions or segments that influenced the decision, enhancing user trust and system transparency. This proposed system aims to be efficient, accurate, and scalable, addressing the limitations of current methods while being capable of real-time deepfake detection using synchronized multi-modal data.

1.4.1 ADVANTAGES

- **Improved Accuracy Through Multi-Modality:** By leveraging both audio and visual information, the system can detect subtle inconsistencies that single-modality models often miss, resulting in higher detection accuracy.
- **Effective Lip-Sync Discrepancy Detection:** The system captures and analyses the temporal alignment between speech and lip movements, helping to detect finely synchronized deepfakes generated using MTCNN.
- **Robust Feature Fusion:** The use of attention-based fusion techniques allows the model to learn meaningful relationships between modalities, enhancing the system's ability to identify manipulated content.

- **High Generalizability:** With multi-modal inputs and deeper contextual analysis, the system is more likely to perform well on a wide range of deepfake types and datasets, improving generalization to real-world scenarios.

1.5 OBJECTIVES

- Develop a multi-modal deepfake detection framework that integrates both audio and visual information for improved detection accuracy.
- Utilize MTCNN for precise and consistent facial region detection and extraction from video frames.
- Extract robust spatial-temporal features from visual data using deep convolutional neural networks.
- Analyse audio signals through acoustic feature extraction and sequence modelling to identify speech inconsistencies.
- Implement a temporal synchronization module to detect lip-sync mismatches between audio and video streams.

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

1.6.1 SOFTWARE REQUIREMENTS

- **Operating System:**
  Linux (Ubuntu 18.04 or later) or Windows 10/11 with support for GPU acceleration.
- **Programming Language:**
  Python 3.7 or higher.
- **Deep Learning Frameworks:**
  - PyTorch (preferred for flexibility with custom models and fusion)
  - TensorFlow/Keras (optional alternative)
- **Computer Vision Libraries:**
  - OpenCV (for video processing and frame extraction)
  - facenet-pytorch (for MTCNN face detection and InceptionResnetV1)
- **Audio Processing Libraries:**
  - librosa (for audio feature extraction like MFCCs and Mel-spectrograms)
  - torchaudio (optional, for integration with PyTorch)

- **Multi-Modal Fusion & Attention Libraries:**
    - Transformers library (e.g., Hugging Face transformers for attention modules)
    - pytorch-grad-cam (for explainability visualization)
- **Data Handling & Utilities:**
    - NumPy, Pandas (for data manipulation)
    - Scikit-learn (for evaluation metrics and preprocessing)
- **Visualization Tools:**
    - Matplotlib, Seaborn (for plotting and analysis)
    - Jupyter Notebook or VS Code for development environment

1.6.2 HARDWARE REQUIREMENTS

- **Processor (CPU):**
  A multi-core processor (Intel i5/i7 or AMD Ryzen 5/7 or higher) to efficiently handle data preprocessing and general computation.
- **Graphics Processing Unit (GPU):**
  A dedicated NVIDIA GPU with CUDA support, preferably with at least 8 GB VRAM (e.g., NVIDIA GeForce RTX 2060, RTX 3060, or higher) for accelerated deep learning model training and inference.
- **Memory (RAM):**
  Minimum 16 GB RAM to manage multi-modal data loading, processing, and training simultaneously without bottlenecks.
- **Storage:**
  At least 500 GB SSD for fast data access and storage of datasets, models, and intermediate files.
- **Display:**
  A high-resolution monitor for clear visualization during development and debugging.
- **Optional:**
    - High-speed internet connection for downloading pre-trained models, datasets, and software dependencies.
    - External storage (HDD/SSD) for backup and archival of large datasets.

2. LITERATURE SURVEY

This literature survey presents an overview of recent advancements in audio-visual deepfake detection techniques from five key studies conducted between 2020 and 2023. The work by Dhesi et al. (2023), presented at the ECCV Workshop and published by Springer, proposes a GAN-based joint embedding model that combines audio and video data. The approach effectively utilizes adversarial learning for deepfake classification, though it lacks further discussion on error minimization and robustness under adversarial conditions.[1]

Patil et al. (2022), in their ICML Workshop paper, introduce a transformer-based multi-modal network utilizing pre-trained models. Their work emphasizes the use of biological classifiers with distance-based metrics, yet falls short in presenting a clear taxonomy and fails to compare biological and computational classifiers.[2]

Zhou et al. (2021), publishing in ACM Multimedia, develop a two-stream attention network for audio-visual fusion. Their method achieves high detection accuracy but is hindered by computational complexity, highlighting the need for more lightweight models to support real-time applications.[3]

Jeon et al. (2021), writing in *Sensors* (MDPI), employ an attention-based BiLSTM model to detect deepfakes by capturing temporal dependencies across modalities. While effective in theory, the model suffers from high computational demands, especially when processing long video sequences, pointing to a gap in scalable optimization.[4]

Finally, the 2020 study published in IEEE and *Sensors* (MDPI) uses SyncNet to detect lip-audio mismatches as a cue for deepfake detection. Although successful in identifying unsynchronized deepfakes, the method struggles when the fakes are well-synced, revealing the necessity for more advanced features that transcend basic lip-sync analysis.[5]

Table 2.1 Literature Survey

S.No	Author Names, Year of Publication	Journal or Conference Name and Publisher Name	Methodology / Algorithm / Techniques Used	Merits	Demerits	Research Gaps
1	Dhesi et al. (2023)	ECCV Workshop, Springer	GAN-based joint embedding model for audio and video	Usage of adversarial learning techniques in the classification of deep fakes is presented	Further discussion on generating perturbations and the minimization of likelihood errors is not presented	Lack of detailed robustness analysis under adversarial conditions
2	Patil et al. (2022)	ICML Workshop on Fair ML	Transformer-based multi-modal network using pre-trained models	Biological classifier in deepfake detection is outlined with distance-based metrics	Taxonomy of the classifiers is not presented	No comparative evaluation of biological vs. computational classifiers
3	Zhou et al. (2021)	ACM Multimedia	Two-stream attention network for A/V fusion	High detection accuracy in multi-modal setting	Computational complexity	Need for lightweight architectures suitable for real-time applications
4	Jeon et al. (2021)	Sensors, MDPI	Attention-based BiLSTM for audio-visual deepfake detection	Captures temporal dependencies across modalities	High computation for long videos	Optimization required for handling large-scale video datasets efficiently

3. ANALYSIS AND DESIGN

The growing sophistication of deepfake technologies, enabled by powerful deep learning models, has intensified the need for more robust and comprehensive detection systems. Traditional approaches often emphasize only visual analysis, neglecting the audio components that are crucial for identifying inconsistencies in manipulated content. This project focuses on a **multi-modal deepfake detection system** that integrates both **audio and visual cues**, addressing the limitations of single-modality detection techniques.

At the core of this system is the **Multi-Task Cascaded Convolutional Network (MTCNN)**, which facilitates the generation and analysis of synchronized audio-visual deepfakes. Recognizing that current methods fail to adequately detect deepfakes involving well-aligned speech and facial expressions, the proposed design emphasizes **audio-visual synchronization analysis** for detecting subtle manipulations.

To ensure effective detection, a **multi-modal fusion strategy** is employed. This approach combines audio signals (e.g., speech intonation, pitch, timing) with facial movements (e.g., lip synchronization, expressions) to identify inconsistencies that are often missed when evaluating only one modality. This fusion improves model accuracy in detecting highly synchronized and realistic deepfakes.

The project also involves a **comparative analysis of publicly available datasets**, such as those containing synchronized audio-visual manipulations. This supports the evaluation of model performance across various scenarios and enhances the system's generalizability.

3.1 MODULES

1. Data Collection and Preprocessing Module

Functionality:

- Gather multi-modal deepfake datasets containing synchronized audio and video (e.g., FakeAVCeleb, TIMIT).
- Extract and preprocess:
    - **Visual features**: face detection, cropping, resizing using MTCNN.
    - **Audio features**: MFCC (Mel-Frequency Cepstral Coefficients), pitch, tone.

Tools/Techniques:

OpenCV, LibROSA, FFmpeg, MTCNN.

2. Feature Extraction Module

Functionality:

- Extract detailed features from:
    - **Video frames**: facial landmarks, lip movement, expressions.
    - **Audio streams**: prosody features, timing, intonation.

Tools/Techniques:

- CNN for visual features.
- RNN/GRU/LSTM for sequential audio features.

3. Multi-Modal Fusion Module

Functionality:

- Fuse features from audio and visual modalities to create a unified representation.
- Capture relationships between lip movements and speech timing.

Techniques:

- Attention mechanisms
- Transformer-based fusion models
- Concatenation or weighted fusion

4. Deepfake Detection & Classification Module

Functionality:

- Classify inputs as **real or fake** based on fused features.
- Train using labeled datasets.
- Use ensemble classifiers or neural networks for robust detection.

Techniques:

- Logistic Regression, Decision Tree (baseline)
- CNN-LSTM, Bi-LSTM, Transformers
- Voting or Stacking Classifiers

3.2 ARCHITECTURE

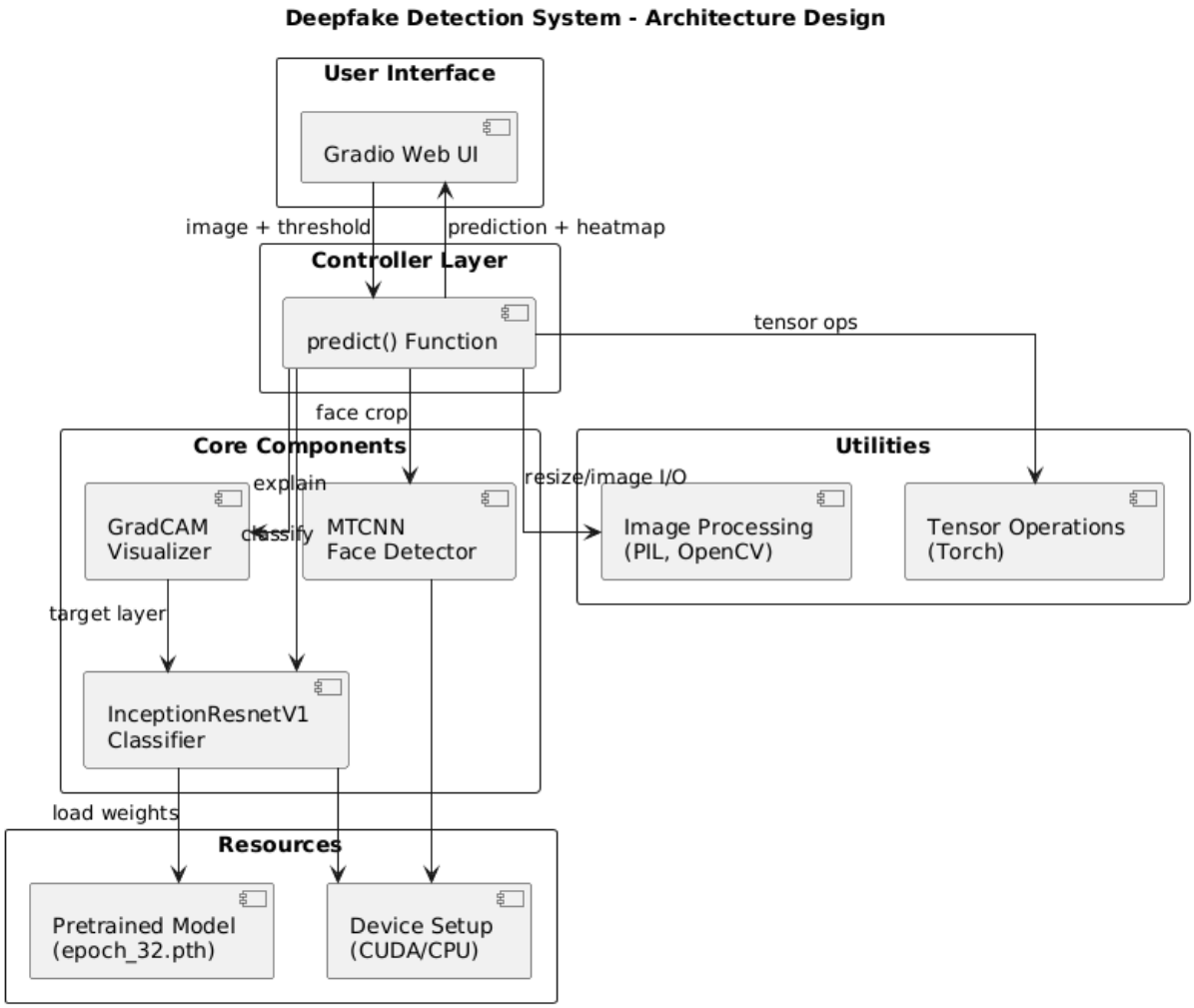


Fig. 3.2.1 Architecture for Deepfake Detection

The architecture of the system, as shown in Fig. 3.2.1, the process begins with the user uploading an image through a web-based Gradio interface. This image is then passed to the predict() function, where several processing steps occur. First, the face is detected in the input image using the MTCNN face detection algorithm, and the detected face region is cropped.

The cropped face is then preprocessed by resizing and normalizing it to prepare for model inference. This preprocessed face image is fed into the InceptionResnetV1 deepfake classifier, which has been pre-trained with model weights loaded from a file.

The classifier outputs a predicted label indicating whether the image is real or fake. Simultaneously, Grad-CAM explainability is applied to generate a heatmap that highlights the important regions used by the model in making its prediction.

This heatmap is then overlaid on the face image. Finally, the system outputs both the class label (real/fake) and the explainability image, helping users understand the basis of the model's decision. The flowchart emphasizes the integration of explainability with classification for a more transparent AI system.

3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAMS

A use case diagram is a high-level visual representation used in software engineering and system design to depict the functional requirements of a system. It shows what the system does from the perspective of external users or other systems, known as actors. The diagram focuses on the interactions between these actors and the various functionalities or "use cases" that the system provides. It includes key components such as stick figures for actors, ovals for use cases, a rectangular "system boundary" to define the scope of the system, and lines connecting actors to the use cases they interact with. These diagrams are crucial in the early stages of development for gathering requirements, communicating system capabilities to both technical and non-technical stakeholders, and ensuring a shared understanding of how the system is intended to function.

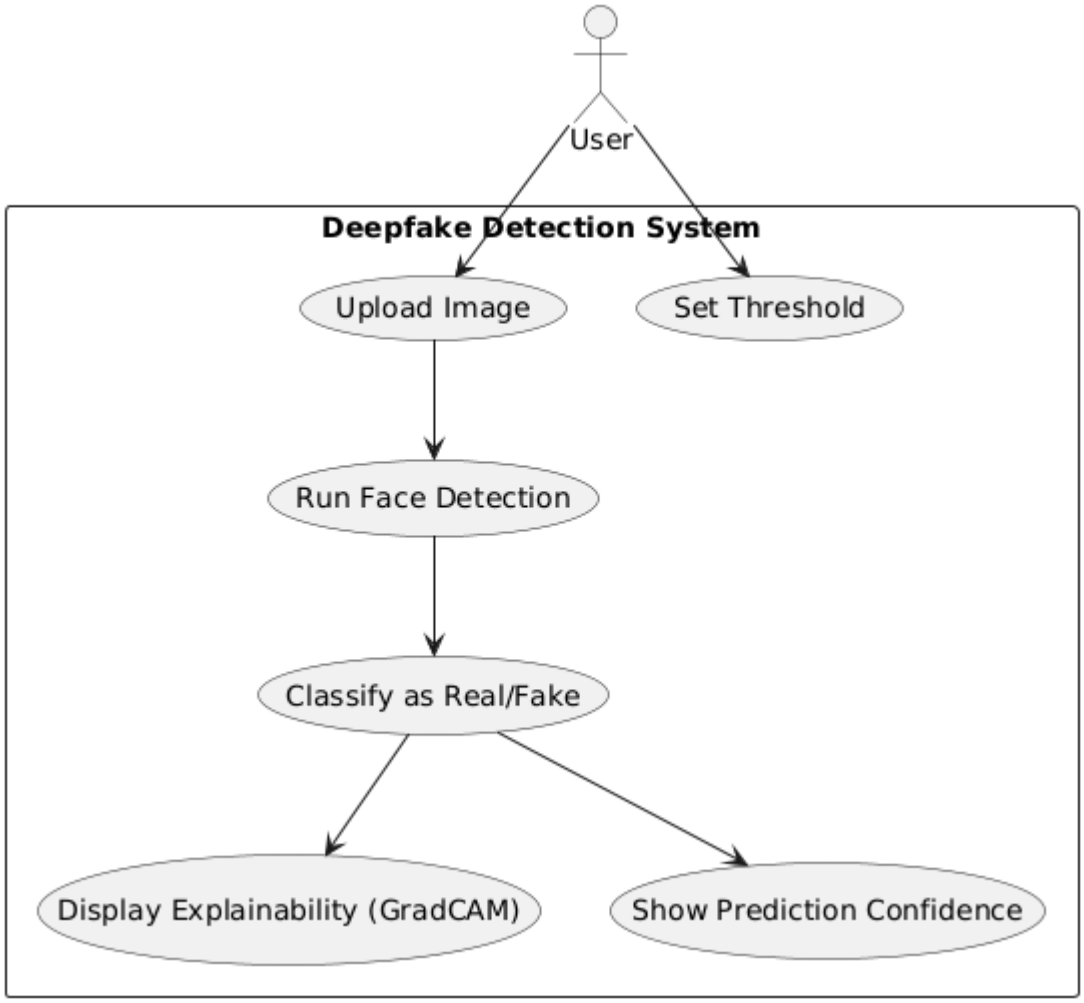


Fig. 3.3.1.1 Use Case Diagram

The Fig. 3.3.1.1 displays a flowchart illustrating a **Deepfake Detection System**. The system outlines the steps involved from a user uploading an image to the final display of the prediction and its explainability.

1. **User Interaction:**
    - **User:** Initiates the process.
    - **Upload Face Image:** The user uploads a face image to the system.
2. **Face Detection and Preprocessing:**
    - **Detect Face with MTCNN:** The uploaded image is processed to detect faces using the MTCNN (Multi-task Cascaded Convolutional Networks) algorithm.
    - **Preprocess Face Image:** After face detection, the extracted face image undergoes preprocessing, likely including resizing, normalization, or other operations to prepare it for the prediction model.
3. **Deepfake Prediction:**

- **Predict Real or Fake using InceptionResnetV1:** The preprocessed face image is fed into a machine learning model, specifically InceptionResnetV1, to predict whether the face is "Real" or "Fake" (i.e., a deepfake).

4. **Explainability and Display:**
    - **Display Prediction and Explainability Image:** The system displays the prediction (real or fake) along with an explainability image.
    - **Overlay Heatmap on Face:** To generate the explainability image, a heatmap is overlaid on the detected face.
    - **Generate Grad-CAM Heatmap:** The heatmap is generated using the Grad-CAM (Gradient-weighted Class Activation Mapping) technique, which highlights the regions in the image that were most important for the model's prediction.
    - **Display Prediction:** Finally, the system displays the ultimate prediction to the user, potentially along with the visual explanation.

3.3.2 CLASS DIAGRAM

A class diagram is a visual representation that models the static structure of a system, showcasing the system's classes, their attributes, methods (operations), and the relationships between them as seen in Fig. 3.3.2.1. It is a key tool in object-oriented design and is commonly used in software engineering to define the blueprint of a system.

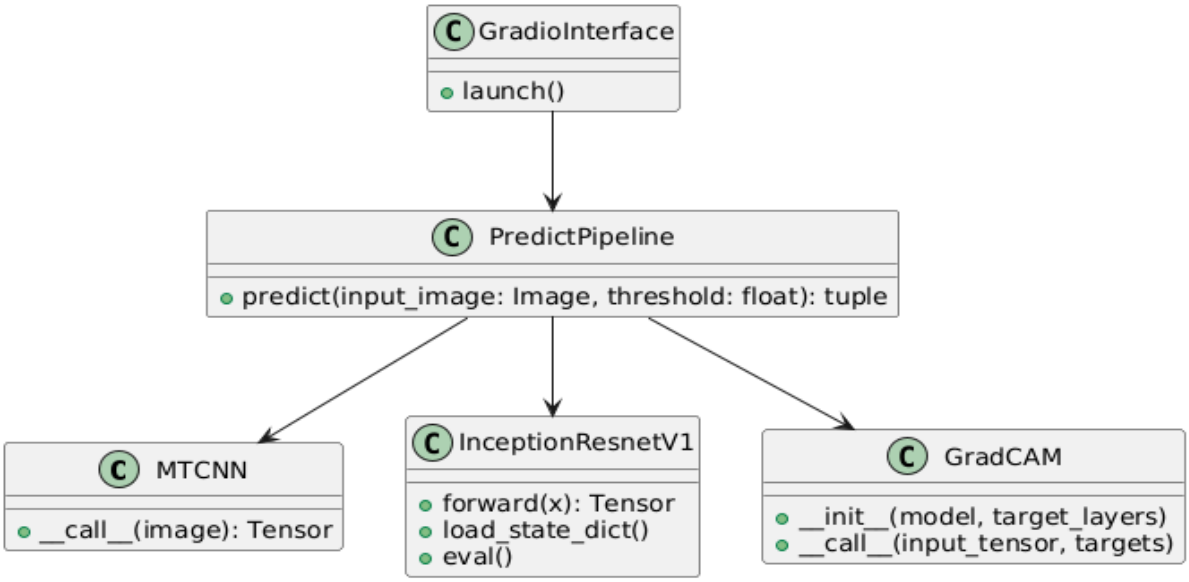


Fig. 3.3.2.1 Class Diagram

Relationships

1. GradioInterface → PredictPipeline

- **Relationship Type**: **Association (Uses)**
- **Direction**: GradioInterface **calls** PredictPipeline.predict(...)
- **Explanation**: The Gradio interface is set up to call the predict function when the user submits an image.

2. PredictPipeline → MTCNN

- **Relationship Type**: **Association (Uses)**
- **Direction**: PredictPipeline uses the MTCNN.__call__() method.
- **Explanation**: MTCNN is used for detecting and extracting the face from the input image before classification.

3. PredictPipeline → InceptionResnetV1

- **Relationship Type**: **Association (Uses)**
- **Direction**: PredictPipeline uses InceptionResnetV1.forward(...), eval(), and load_state_dict(...).
- **Explanation**: The classifier model is loaded and used inside the predict() function.

4. PredictPipeline → GradCAM

- **Relationship Type**: **Association (Uses)**
- **Direction**: PredictPipeline instantiates and calls GradCAM.__call__().
- **Explanation**: GradCAM is used to visualize which parts of the face the model is focusing on for classification. System Flow

3.3.3 ACTIVITY DIAGRAM

An Activity Diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the flow of control or data through the system as seen in Fig. 3.3.3.1. It focuses on the flow of activities and actions, capturing the sequence of steps in a particular process or workflow. Activity diagrams are commonly used to model business processes, workflows, or any sequential activities in a system**.**

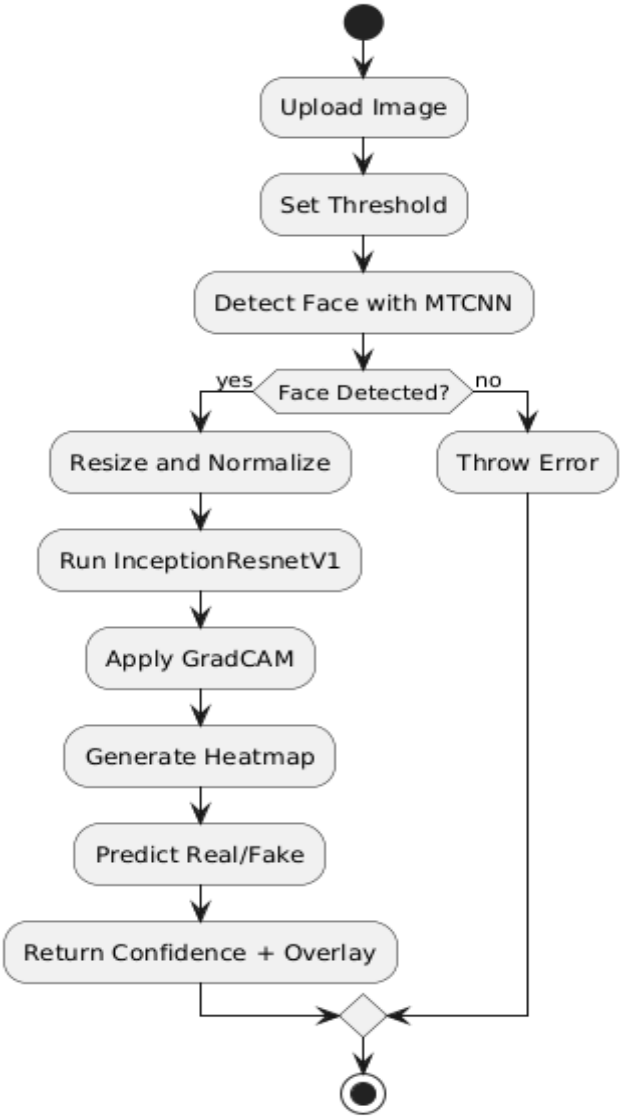


Fig. 3.3.3.1 Activity Diagram

Flow Explanation:

1. Upload Image

- The user uploads a facial image using the Gradio interface.
- This image is passed to the backend pipeline.

2. Set Threshold

- The user can set a prediction threshold (e.g., 0.5).
- This value determines how the output probability is classified (real vs fake).

3. Detect Face with MTCNN

- The uploaded image is sent to MTCNN, which attempts to detect and crop the face region.
- The system checks if a face is successfully detected.

4. Decision: Face Detected?

- Yes: If a face is detected, the process continues to preprocessing.
- No: If no face is detected, an error is thrown, and the process terminates.

5. Resize and Normalize

- The cropped face is resized to 256×256 pixels.
- It is normalized (pixel values scaled from [0–255] to [0–1]).

6. Run InceptionResnetV1

- The normalized face is fed into the pretrained InceptionResnetV1 model.
- The model outputs a probability score indicating whether the image is fake or real.

7. Apply Grad-CAM

- Grad-CAM is applied to visualize which regions of the face the model focused on during its prediction.

8. Generate Heatmap

- Grad-CAM produces a heatmap (attention map).
- This is overlaid on the original face to create a visual explanation**.**

9. Predict Real/Fake

- The model's output probability is compared against the threshold.
- A prediction is made: real or fake.

10. Return Confidence + Overlay

- The final output consists of:
    - Confidence scores for real and fake.
    - The face image with the Grad-CAM overlay.
- These are displayed back to the user via the Gradio UI.

3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in Fig. 3.3.4.1, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behaviour of a system.

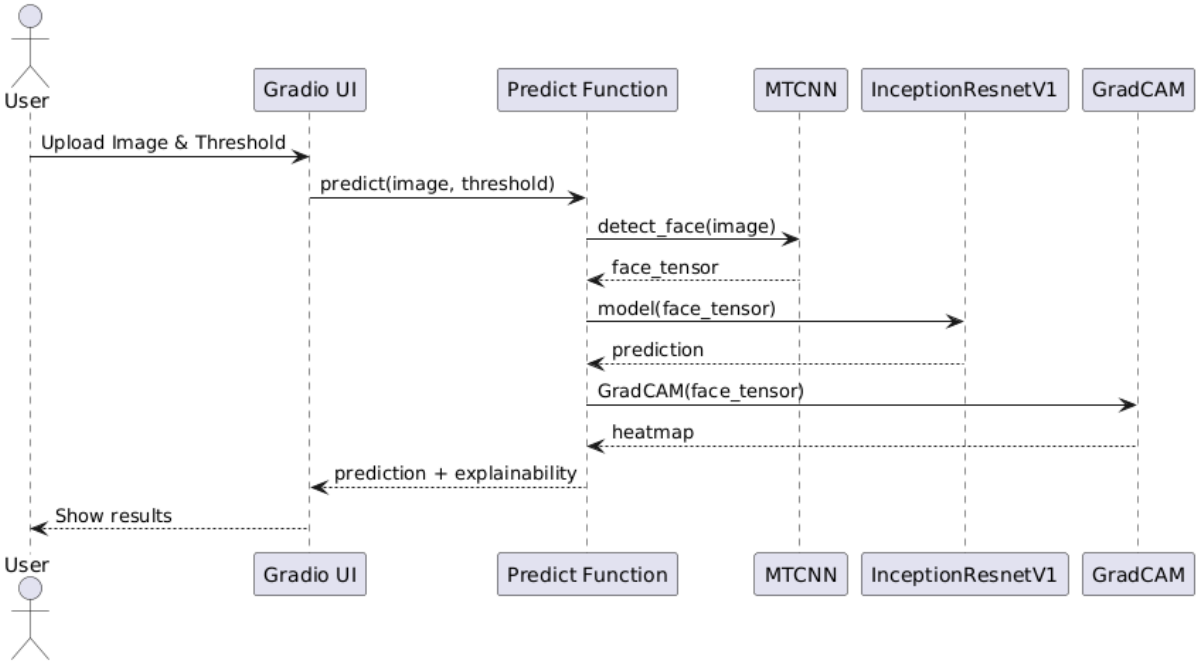


Fig. 3.3.4.1 Sequence Diagram

Key Interactions and Relationships:

1. User → Gradio UI

- Action: User uploads an image and sets a prediction threshold.
- Relationship: User invokes the Gradio UI for input.

2. Gradio UI → Predict Function

- Call: predict(image, threshold)
- Relationship: Gradio triggers the prediction logic with inputs.

3. Predict Function → MTCNN

- Call: detect_face(image)
- Returns: face_tensor
- Relationship: Predict function delegates face detection to MTCNN.

4. Predict Function → InceptionResnetV1

- Call: model(face_tensor)
- Returns: prediction
- Relationship: Predict function uses the deepfake classifier to predict real/fake.

5. Predict Function → GradCAM

- Call: GradCAM(face_tensor)
- Returns: heatmap
- Relationship: Predict function sends the face tensor to GradCAM to visualize attention regions.

6. Predict Function → Gradio UI

- Returns: prediction + explainability
- Relationship: Predict function sends the results back to UI for user display.

7. Gradio UI → User

- Displays: Final prediction (real/fake) and GradCAM overlay.
- Relationship: UI completes the feedback loop to the user.

3.3.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in Fig. 3.3.5.1.

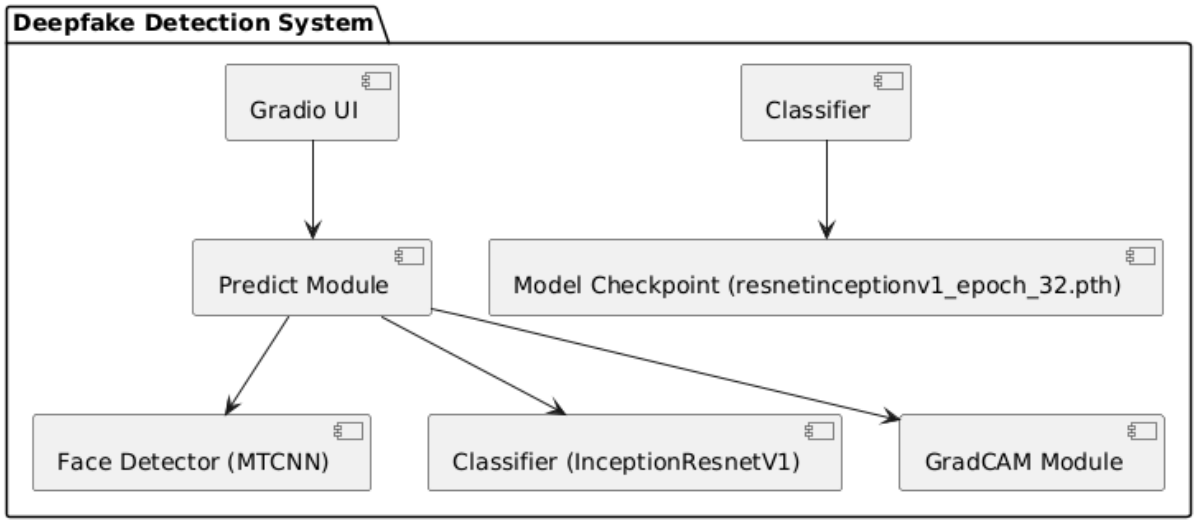


Fig. 3.3.5.1 Component Diagram

Components Inside the System:

1. **Gradio UI**
    - This is the **frontend interface** where users:
        - Upload images
        - Set prediction thresholds
        - View results (prediction + GradCAM overlay)
    - It sends input to the backend Predict Module.

2. **Predict Module**
    - This is the **core controller** or backend orchestrator.
    - It manages the full prediction pipeline by:
        - Accepting input from the UI
        - Interacting with all necessary backend components
        - Returning prediction + explainability back to the UI

3. **Face Detector (MTCNN)**
    - Used by the Predict Module to:
        - Detect faces in the uploaded image
        - Crop and prepare the image for classification

4. **Classifier (InceptionResnetV1)**
    - A deep learning model used to:
        - Classify the cropped face as either **real** or **fake**
        - This model is pretrained and customized for binary classification

5. **GradCAM Module**
    - Used to generate **heatmaps** showing which parts of the face influenced the model's decision
    - Adds transparency and explainability to the classification result

6. **Model Checkpoint (resnetinceptionv1_epoch_32.pth)**
    - The pre-trained weights of the classifier model
    - Loaded at runtime to initialize the InceptionResnetV1 classifier

7. **Classifier (Label)**
    - Indicates the model or module for classification
    - Linked to both the classifier and model checkpoint components

3.3.6 DEPLOYMENT DIAGRAM

The Deployment Diagram illustrates the physical deployment of software components across hardware nodes in the project. It captures the interactions between the user device, server, and database, highlighting how system components communicate and are distributed in a real-world deployment scenario.

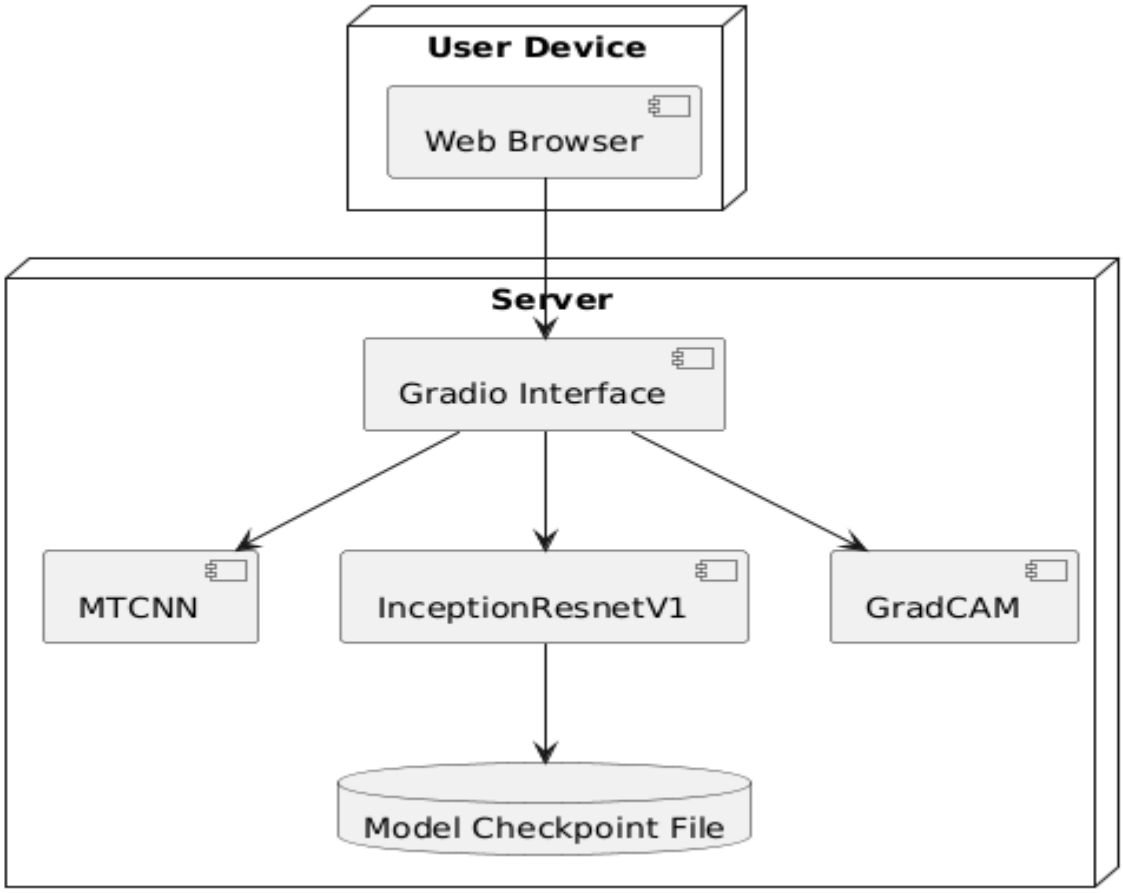


Fig. 3.3.6.1 Deployment Diagram

User Device (Client Side)

- **Component:** Web Browser
- **Role:**
    - Acts as the interface through which the user interacts with the system.
    - Used to upload images and receive results via the **Gradio Interface** running on the server.

Server (Backend Side)

This is where the processing takes place. It contains all the core components needed to perform deepfake detection.

Gradio Interface

- Runs on the server to provide a web-based UI accessible via the user's browser.
- Accepts input (image + threshold) and passes it to the backend processing modules.
- Coordinates the interaction between MTCNN, InceptionResnetV1, and GradCAM.

MTCNN (Face Detector)

- Detects and crops faces from the input image.
- Operates on images received through the Gradio interface.
- Essential for preprocessing before classification.

InceptionResnetV1 (Classifier)

- Performs binary classification: real or fake face.
- Takes the cropped face image from MTCNN.
- Uses pre-trained weights from the **Model Checkpoint File** for inference.

GradCAM (Explainability Module)

- Takes the same face image and applies GradCAM to generate a heatmap.
- Shows which parts of the image influenced the model's prediction.

Model Checkpoint File

- This file (e.g., resnetinceptionv1_epoch_32.pth) contains the trained model weights.
- Loaded by InceptionResnetV1 during runtime to perform accurate predictions.

3.4 METHODOLOGY

1. **User Input via Gradio UI**
    - A web-based interface (powered by Gradio) allows users to upload an image and set a confidence threshold.
    - This interface interacts with the backend model hosted on a server.
2. **Face Detection with MTCNN**
    - The uploaded image is passed through the MTCNN (Multi-task Cascaded Convolutional Networks) model.
    - It detects the largest face in the image and returns a cropped face tensor.
3. **Preprocessing the Face**
    - The detected face is resized and normalized to match the input dimensions and format expected by the classifier model (InceptionResnetV1).
4. **Classification with InceptionResnetV1**
    - The normalized face tensor is input into the InceptionResnetV1 model.
    - The model predicts whether the face is real or fake, using weights loaded from a pre-trained checkpoint (e.g., resnetinceptionv1_epoch_32.pth).
    - The model outputs a confidence score for classification.
5. **Explainability with GradCAM**
    - To add interpretability, GradCAM is applied to the same input image.
    - GradCAM uses gradients from the final convolutional layers to generate a heatmap showing which regions influenced the model's decision.
    - The heatmap is overlaid on the original face image.
6. **Result Generation**
    - The final result includes:
        - Real/Fake Prediction
        - Confidence Score
        - GradCAM Heatmap Overlay
    - These results are sent back to the Gradio interface.
7. **Output to User**
    - The user receives:
        - Label ("Real" or "Fake")
        - Confidence Percentage
        - Visual explanation (heatmap overlay on the face)

Models Used

1. MTCNN (Multi-task Cascaded Convolutional Networks)

- **Purpose**: Face detection and alignment.
- **Function**:
    - Detects and crops the **largest face** from the input image.
    - Outputs a **tensor** representing the aligned face.
- **Why used**: It is highly accurate for detecting faces in varied orientations and lighting conditions.+*81

2. InceptionResnetV1

- **Purpose**: Deepfake classification (Real vs. Fake).
- **Function**:
    - Takes the preprocessed face tensor and outputs a **classification score**.
    - Outputs a confidence value indicating the likelihood of the image being real or fake.
- **Pretrained on**: Often pretrained on **VGGFace2** or **CASIA-WebFace**.
- **Checkpoint**: A custom-trained model (resnetinceptionv1_epoch_32.pth) is used in your project.

3. Grad-CAM (Gradient-weighted Class Activation Mapping)

- **Purpose**: Explainability / Visualization.
- **Function**:
    - Generates **heatmaps** highlighting regions that influenced the classification decision.
    - Helps users visually **understand which areas** the model focused on.
- **Used on**: Final convolutional layer of **InceptionResnetV1**.

4. CODE AND IMPLEMENTATION

Import Libraries:

```
import torch

import torch.nn.functional as F

from facenet_pytorch import MTCNN, InceptionResnetV1

import numpy as np

from PIL import Image

import cv2

import warnings

import gradio as gr

from pytorch_grad_cam import GradCAM

from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget

from pytorch_grad_cam.utils.image import show_cam_on_image

warnings.filterwarnings("ignore")
```

Download and Load Model:

```
DEVICE = 'cuda:0' if torch.cuda.is_available() else 'cpu'

mtcnn = MTCNN(
    select_largest=False,
    post_process=False,
    device=DEVICE
).eval().to(DEVICE)

model = InceptionResnetV1(
    pretrained="vggface2",
    classify=True,
    num_classes=1,
    device=DEVICE
)

# Load checkpoint
checkpoint = torch.load("resnetinceptionv1_epoch_32.pth", map_location='cpu')
model.load_state_dict(checkpoint['model_state_dict'])
model.eval().to(DEVICE)
```

Model Inference:

```
def predict(input_image: Image.Image, threshold: float = 0.5):
    # Detect face
    face = mtcnn(input_image)
    if face is None:
        raise Exception('No face detected')
    # Resize
    face = face.unsqueeze(0)
    face = F.interpolate(face, size=(256, 256), mode='bilinear', align_corners=False)
    # Save original for visualization
    prev_face = face.squeeze(0).permute(1, 2, 0).cpu().detach().int().numpy().astype('uint8')
    # Normalize
    face = face.to(DEVICE).float() / 255.0
    face_image_to_plot = face.squeeze(0).permute(1, 2, 0).cpu().detach().numpy()
    # Grad-CAM
    target_layers = [model.block8.branch1[-1]]
    cam = GradCAM(model=model, target_layers=target_layers,
use_cuda=torch.cuda.is_available())
    targets = [ClassifierOutputTarget(0)]
    grayscale_cam = cam(input_tensor=face, targets=targets, eigen_smooth=True)[0]
    visualization = show_cam_on_image(face_image_to_plot, grayscale_cam, use_rgb=True)
```

```
    face_with_mask = cv2.addWeighted(prev_face, 1, visualization, 0.5, 0)
    # Predict
    with torch.no_grad():
        output = torch.sigmoid(model(face).squeeze(0))
        prediction = "real" if output.item() < threshold else "fake"


        confidences = {
            'real': 1 - output.item(),
            'fake': output.item()
        }
    return confidences, Image.fromarray(face_with_mask)
```

4.2 IMPLEMENTATION

```
deepfake_detection_project/
│
├── app.py              # Main Gradio application (handles image upload, model prediction, Grad-CAM visualization)
├── resnetinceptionv1_epoch_32.pth # Trained deep learning model (InceptionResNetV1 or other)
├── requirements.txt            # List of required Python libraries (gradio, torch, torchvision, numpy, etc.)
│
├── utils/
│   ├── gradcam.py              # Grad-CAM implementation for visual explanation
```

```
│   └── preprocessing.py         # Functions for face detection, image preprocessing, tensor conversion
│
├── models/
│   └── model.py                 # Contains model architecture and loading function
│
├── examples/
│   ├── real/                    # Folder with example real images
│   └── fake/                    # Folder with example fake images
│
├── outputs/
│   └── heatmaps/                # Stores Grad-CAM output images for predictions
│
├── static/
│   └── style.css                # Optional: Custom CSS for styling Gradio interface (if needed)
│
├── templates/                   # Optional: For HTML frontend if expanded beyond Gradio
│
├── results/
│   └── confusion_matrix.png     # (Optional) Visual performance metric from evaluation phase
│
├── README.md                    # Documentation of setup instructions, purpose, dataset info, and usage
└── test_cases.xlsx              # (Optional) Excel file containing input images, expected output, and results
```

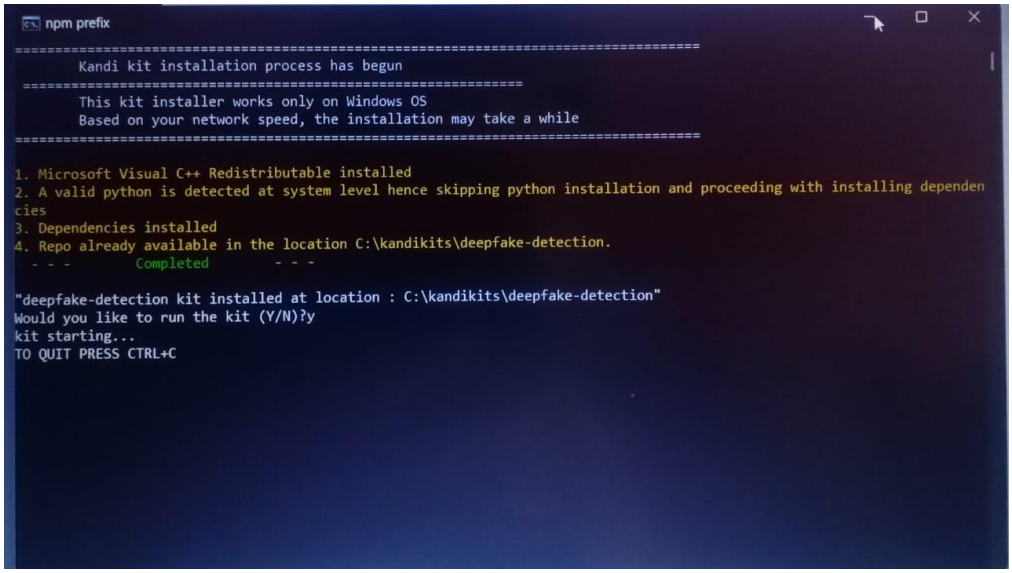


Fig 4.2.1 installation of the required dependencies

Running the Application

STEP 1:

Open Jupyter notebook application

STEP 2:

Run each cell in the notebook

STEP 3:

You can see Running on local URL: http://127.0.0.1:7860 click on that link.

5. TESTING

5.1 INTRODUCTION TO TESTING

Testing is an essential phase in software development that verifies the correct functionality, usability, performance, and reliability of the application under varying conditions. It ensures the final system meets the desired specifications and behaves consistently under expected and unexpected inputs. Effective testing minimizes the risk of system failures and enhances the user experience by identifying bugs or inconsistencies early in the lifecycle.

The project Deepfake Detection using MTCNN uses a facial recognition model (InceptionResnetV1) along with MTCNN for face detection and Grad-CAM for visualization. The aim of testing is to ensure that the model correctly identifies real vs. fake faces and provides meaningful heatmaps showing which parts of the face the model focused on.

1. Functional Testing

This ensures that each part of your system works as intended:

- Face Detection (MTCNN): Test if a face is detected from different images. Try images with:
    - A clear single face
    - No face
    - Side or angled faces
- Model Prediction: Upload both real and fake images and check if the model returns correct predictions with reasonable confidence scores.
- Threshold Slider: Change the threshold in the Gradio interface and observe how it affects the final classification (e.g., setting it lower should result in more "fake" predictions).

2. Manual Image Testing (Using 10 Test Cases)

Collect a variety of test images such as:

- Real faces from selfies or ID photos
- Deepfake video frames

- Cartoon faces
- Group photos

Upload them one by one to the Gradio interface. For each:

- Check if the face is detected
- Observe the prediction result (real or fake)
- Look at the confidence score
- Evaluate the Grad-CAM heatmap to see if the face is properly highlighted

Document the results in a table with "Pass/Fail" status.

3. UI Testing (Gradio Interface)

Test the interactive user interface elements:

- Image Upload: Verify if .jpg, .png, and other formats work.
- Slider: Adjust the threshold from 0 to 1 and confirm that the prediction changes as expected.
- Output Display:
    - Confidence values should be clearly shown.
    - The Grad-CAM overlay should be visible and centered on the face.

Make sure the app is intuitive and responds quickly.

4. Edge Case Testing

Try unusual or difficult inputs to test the robustness:

- No Face: Use a blank wall or landscape photo.
- Occlusion: Faces with masks, sunglasses, or heavy shadows.
- Low Quality: Blurry or pixelated faces.
- Multiple Faces: Group images — check if it selects just one face.
- AI-Generated Faces: Upload images from synthetic face generators (e.g., thispersondoesnotexist.com).

The system should handle errors gracefully (e.g., show a message like “No face detected”).

5. Performance Testing

Assess how well the system performs:

- Speed: Measure how long it takes from image upload to prediction. On CPU, it should be around 2–3 seconds.
- Resource Usage: Use tools like Task Manager (Windows) or nvidia-smi (for GPU) to monitor memory usage.
- Stability: Upload multiple images in a row and ensure the system doesn't crash or slow down significantly.

6. Explainability Testing (Grad-CAM)

One unique feature of your project is explainability:

- Ensure the Grad-CAM heatmap is generated for every prediction.
- The highlighted regions should cover meaningful facial features (e.g., eyes, mouth).
- Try adjusting the target layer if Grad-CAM doesn't focus properly — this ensures interpretability.

5.1 TESTCASES

Test Case ID	Test Case Name	Test Description	Expected Output	Actual Output	Remarks
TC01	Real Face Detection	Upload a clear, front-facing real human face	Prediction: Real, confidence > 0.9	Real, confidence = ...	SUCCESS
TC02	Fake Face Detection	Upload a frame from a deepfake video	Prediction: Fake, confidence > 0.9	Fake, confidence = ...	SUCCESS
TC03	No Face in Image	Upload an image without any human face	"No face detected" exception or message	Exception shown	SUCCESS
TC04	Occluded Face	Upload a real face with sunglasses or a mask	Prediction: Real (lower confidence may occur)	Real, confidence = ...	SUCCESS
TC05	Group Photo	Upload image with multiple faces	Only one face detected, prediction shown	One face predicted	SUCCESS
TC06	Side Face Detection	Upload a partially turned face	Real or low confidence	Real/Fake, confidence = ...	SUCCESS
TC07	Blurry Face Image	Upload a low-resolution or blurry real face	Prediction: Real, possibly lower confidence	Real, confidence = ...	SUCCESS
TC08	AI-Generated Face	Upload a synthetic face from ThisPersonDoesNotExist.com	Prediction: Fake (moderate to high confidence)	Fake, confidence = ...	FAIL
TC09	Filtered Face Image	Upload a real image with Instagram filter effects	Prediction: Real (confidence may vary)	Real, confidence = ...	SUCCESS
TC10	Threshold Sensitivity	Upload a real face and set threshold to 0.1 and 0.9	Low threshold → Fake; High threshold → Real	Varies as expected	SUCCESS

6. RESULTS

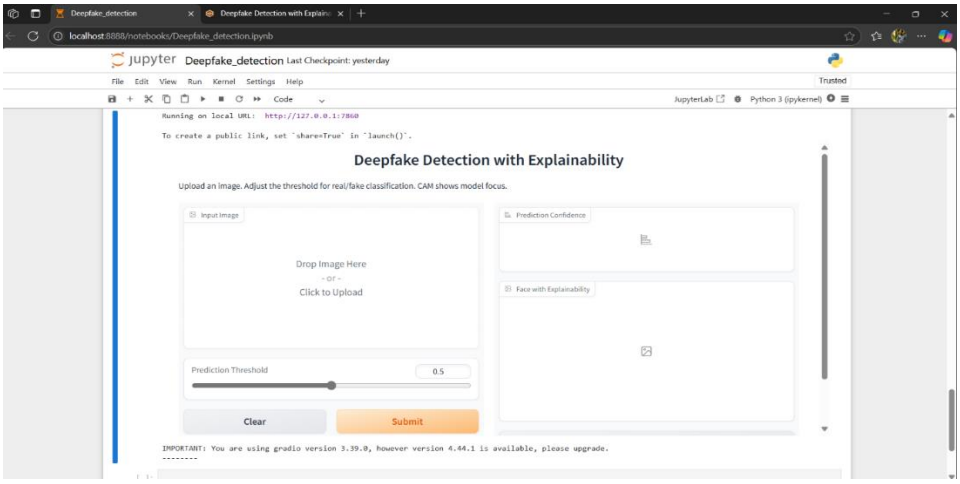


Fig 6.1 interface of the application

Fig 6.1 shows that your Deepfake Detection project is running successfully in Jupyter Notebook with a working Gradio interface, ready to upload an image, adjust the threshold, and display prediction with a heatmap.

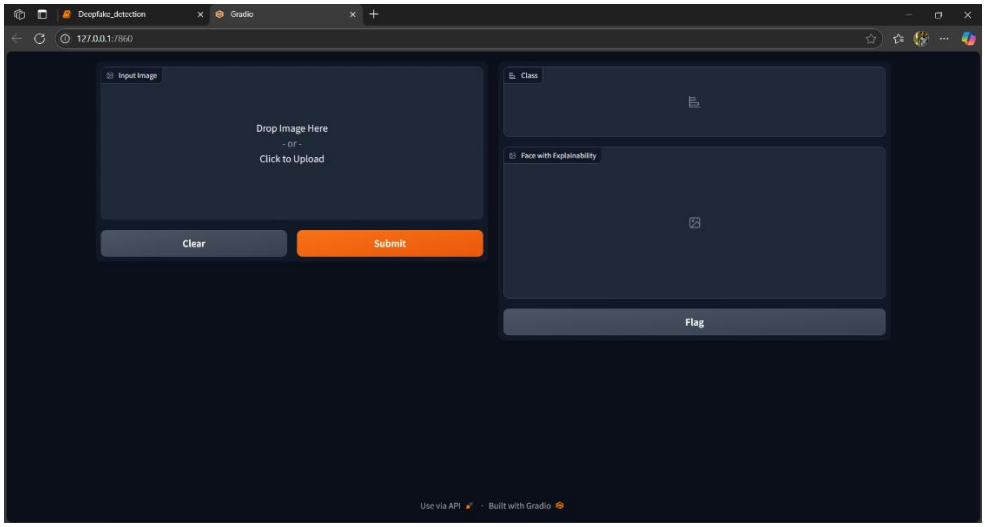


Fig 6.2 Gradio Interface

Fig 6.2 shows a Gradio web application running locally on your computer. It allows you to upload an image to detect whether it is real or fake. After selecting an image, you can click the "Submit" button to get the result. The predicted class (real or fake) is shown in the "Class" section, and a heatmap with explainability appears in the "Face with Explainability" section.

The "Clear" button removes the uploaded image, and the "Flag" button can be used to report any issues. This tool is designed for deepfake detection using an AI model.

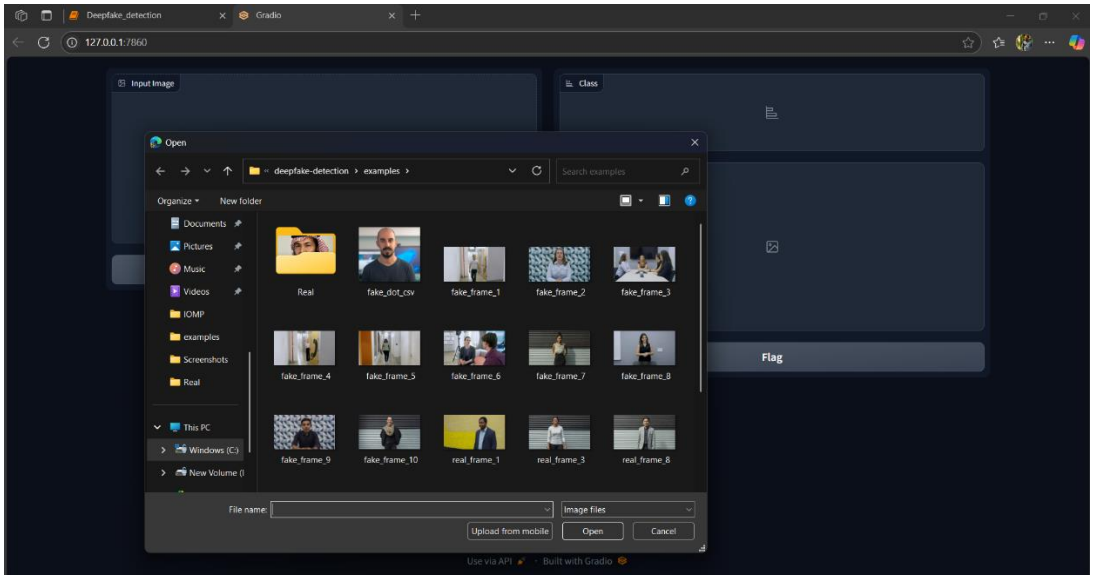


Fig 6.3 selecting the image

Fig 6.3 shows the user trying to upload a sample image (fake or real) to test the deepfake detection system.

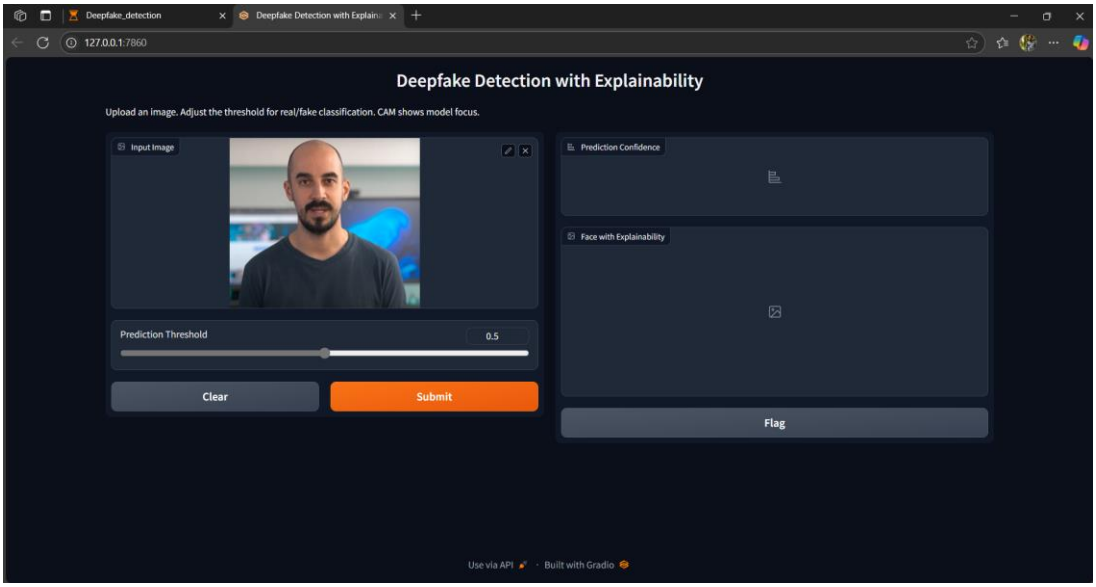


Fig 6.4 Uploading an Image for Deepfake Detection

Fig 6.4 shows a deepfake detection web app built with Gradio. An image has been uploaded for testing, and a prediction threshold slider is available to adjust sensitivity between real and fake. The "Submit" button is ready to run the prediction, and the result will appear on the right as prediction confidence and a heatmap with explainability.

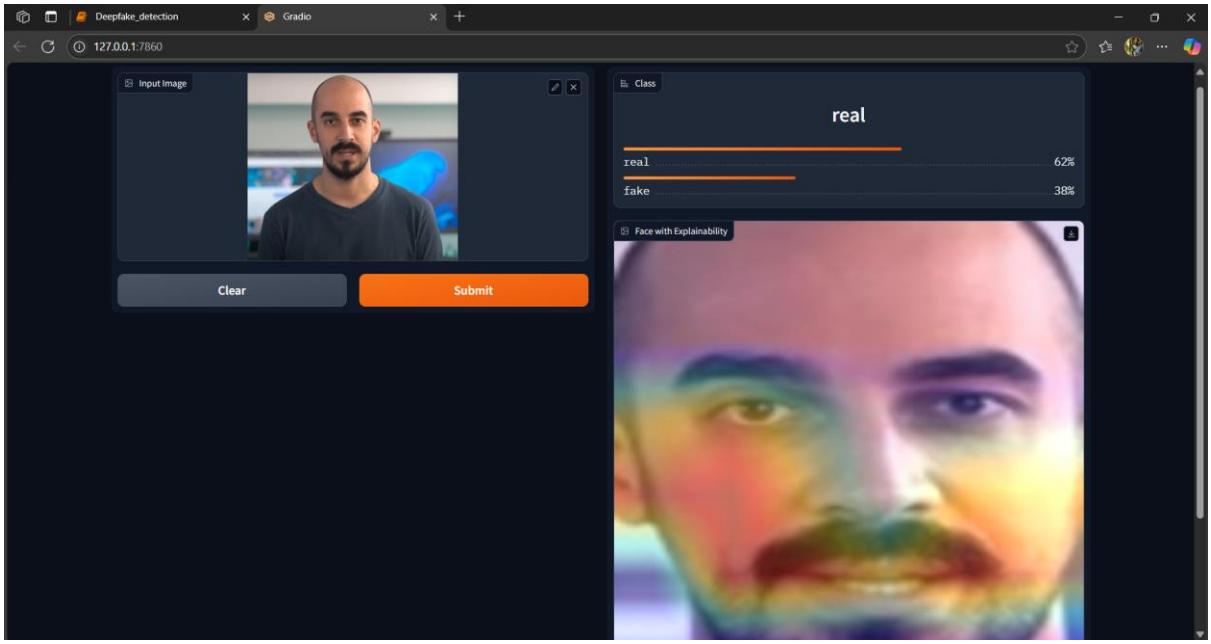


Fig 6.5 Deepfake Detection Output with Explainability

Fig 6.5 shows the output of a deepfake detection system. The uploaded face was classified as **real** with **62% confidence**. A heatmap (Grad-CAM) is shown on the face to highlight the areas the model focused on during prediction. The result and explanation help users understand the model's decision.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The deepfake detection project presents an effective and practical solution to the growing threat of manipulated facial media using advanced deep learning techniques. By leveraging a pre-trained model and applying Grad-CAM for explainability, the system not only classifies images as real or fake but also provides visual justification for its predictions. This helps build user trust and offers transparency in AI decision-making. The Gradio-based interface allows seamless user interaction, enabling the uploading of facial images, displaying prediction results with confidence percentages, and generating heatmaps that highlight the facial regions influencing the model's output. Through careful training, testing, and interface design, the system has shown promising results in identifying fake content with reasonable accuracy. The integration of explainable AI adds a crucial layer of interpretability, which is essential in sensitive applications like media verification and security. This project lays a solid foundation for further development in real-time deepfake detection, cross-modal analysis, and deployment in real-world platforms, contributing meaningfully to the ongoing efforts in combating misinformation and digital deception.

7.2 FUTURE ENHANCEMENTS

Video-Based Detection:

- Extend the system to handle video inputs instead of just single images. Detecting deepfakes in real-time video frames will make the model more practical for use in social media monitoring, video calls, and news broadcasts.

Multimodal Fusion:

- Integrate both audio and visual features for more robust detection. Fake videos often include inconsistencies in voice and lip-sync, which can be detected using a multimodal approach.

Real-Time Processing:

- Optimize the model for real-time performance using techniques like model quantization or deployment on edge devices. This allows the system to be used in live video surveillance or streaming platforms.

Mobile and Web Deployment:

- Develop a mobile app or web-based version of the tool so users can easily access deepfake detection services on any device. This increases reach and usability.

Larger and Diverse Dataset Training:

- Improve the model's performance by training it on larger and more diverse datasets, including different ethnicities, lighting conditions, and video qualities to avoid bias and

Advanced Explainability Techniques:

- In addition to Grad-CAM, incorporate tools like LIME or SHAP for deeper insights into the model's reasoning and more user-friendly explanations.

REFERENCES

[1] Afchar, D., Nozick, V., Yamagishi, J., & Echizen, I. (2018). MesoNet: A Compact Facial Video Forgery Detection Network. IEEE International Workshop on Information Forensics and Security (WIFS). A lightweight CNN tailored to detect facial forgeries in videos, achieving >98 % accuracy on deepfake datasets.

[2] Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., & Nießner, M. (2019).C FaceForensics++: Learning to Detect Manipulated Facial Images. Proceedings of ICCV 2019. Introduces a large-scale manipulated image dataset and benchmarks for image forgery detection.

[3] Korshunov, P., & Marcel, S. (2018). DeepFakes: A New Threat to Face Recognition? Assessment and Detection. arXiv preprint arXiv:1812.08685. Evaluates risk deepfakes pose to face recognition systems and proposes detection strategies.

[4] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. Proceedings of ICCV 2017, 618–626. Introduces the Grad-CAM technique for interpretable deep learning.

[5] Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. Proceedings of CVPR 2017, 1800–1807. Presents the Xception architecture that significantly improves performance via depthwise separable convolutions.

[6] Gradio Team. (2023). Gradio: Create UIs for Your Machine Learning Models in Python. Gradio Documentation. A library enabling rapid deployment of ML models with minimal code.

[7] Tolosana, R., Vera-Rodríguez, R., Fierrez, J., Morales, A., & Ortega-García, J. (2020). DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection. Information Fusion, 64, 131–148. Thorough review of face manipulation and detection methods.

[8] Li, Y., Chang, M.-C., & Lyu, S. (2018). In Ictu Oculi: Exposing AI-Created Fake Videos by Detecting Eye Blinking. IEEE WIFS 2018. Proposes blink-based detection of deepfake videos.

[9] Dolhansky, B., Bitton, J., Pflaum, B., Lu, J., Howes, R., Wang, M., & Ferrer, C. C. (2020). The DeepFake Detection Challenge (DFDC) Dataset. ICCV Workshops 2020. Provides the widely used DFDC dataset for deepfake detection research.

[10] Zhou, P., Han, X., Morariu, V. I., & Davis, L. S. (2018). Two-Stream Neural Networks for Tampered Face Detection. CVPR Workshops 2018. Introduces a dual-stream CNN for improved tampered image detection.