Crowd Emotion Recognition

1. Introduction

1.1 Introduction

In today's world, understanding human emotions plays a crucial role in various domains such as psychology, marketing, and human-computer interaction. Emotions significantly impact human decision-making, behavior, and interactions. As technology advances, there is a growing interest in developing systems that can automatically recognize and interpret human emotions. This project focuses on crowd emotion recognition, which involves detecting and classifying the emotions of individuals in a crowd from images and videos.

1.2 Project Overview

The Crowd Emotion Recognition project aims to develop a system that can automatically classify the emotions present within a crowd image or video. The system leverages deep learning techniques and utilizes a pre-trained ResNet50 model to accurately detect and classify emotions. The project integrates a web application built with Streamlit, allowing users to upload or capture images and videos, which are then processed to detect emotions. Various visualizations are provided to help users understand the distribution and performance of the model.

The system is designed to handle grayscale images from the FER2013 dataset, which contains 35,887 facial images categorized into seven emotion classes: angry, disgust, fear, happy, sad, surprise, and neutral. Data augmentation techniques such as rotation, zoom, and horizontal flip are applied to increase the diversity of the training data and improve the model's generalization.

1.3 Problem Statement

Emotion recognition is a complex task due to the subtle and varying expressions of human emotions. Recognizing emotions in a crowd adds another layer of complexity, as it involves detecting multiple faces with different expressions in a single image or video frame. Traditional methods often struggle with accuracy and efficiency, particularly in real-time applications. The challenge is to develop a robust system that can accurately detect and classify emotions in a crowd setting, providing real-time feedback and insights.

1.4 Objectives

The primary objectives of the Crowd Emotion Recognition project are:

1. To develop a machine learning model capable of accurately recognizing and classifying emotions in images and videos.
2. To create a user-friendly web application that allows users to upload or capture media for emotion detection.
3. To implement data augmentation techniques to improve the model's performance and generalization.

4. To provide various data visualizations to help users understand the distribution of emotions and the model's performance.
5. To evaluate the system's performance using metrics such as accuracy, confusion matrix, and ROC curves.

1.5 Study Limitations

While the Crowd Emotion Recognition project aims to achieve accurate and efficient emotion detection, there are several limitations to consider:

1. **Dataset Limitation**: The system is trained on the FER2013 dataset, which may not encompass the full range of human emotions and expressions. This can limit the model's ability to generalize to real-world scenarios with diverse and complex emotions.
2. **Model Limitations**: The ResNet50 model, while powerful, may not capture all the nuances of facial expressions, leading to potential misclassifications.
3. **Real-time Processing**: Although the system aims to provide real-time emotion detection, processing multiple faces in high-resolution videos can be computationally intensive and may result in latency.
4. **Environmental Factors**: Variations in lighting, occlusions, and background noise in images and videos can affect the accuracy of emotion detection.
5. **Ethical Considerations**: The deployment of emotion recognition systems raises ethical concerns regarding privacy and consent, which must be carefully addressed in practical applications.

2. Requirements Specification

2.1 Functional Requirements

System Overview: The Crowd Emotion Recognition system is designed to automatically detect and classify emotions from images and videos. It utilizes a pre-trained ResNet50 model integrated into a Streamlit web application, allowing users to upload media files or capture images and videos using a webcam.

Functional Requirements:

1. **User Interface:**
   - The system must provide a user-friendly interface for uploading images and videos.
   - Users should be able to capture images and videos using a connected webcam.
   - The interface should display the uploaded or captured media along with the detected emotions.
2. **Emotion Detection:**
   - The system must process the uploaded or captured media to detect faces and classify the emotions of each detected face.
   - The system should support detecting multiple faces in a single image or video frame and classify each detected face's emotion.
3. **Media Processing:**

- The system must handle various image formats, including JPG, JPEG, and PNG.
- The system should support video formats such as MP4, AVI, and MOV.
- The system must resize, normalize, and preprocess images before feeding them into the model for emotion detection.

4. **Model Integration:**
   - The system must integrate a pre-trained ResNet50 model for emotion classification.
   - The model should be loaded and initialized upon starting the application to ensure readiness for processing media files.
5. **Result Display:**
   - The system must display the detected emotions for each face in the uploaded or captured media.
   - The system should provide visual feedback, such as bounding boxes around detected faces and emotion labels.
6. **Data Visualization:**
   - The system must include a dashboard displaying various data visualizations, such as emotion distribution, training accuracy, and confusion matrix.
   - Visualizations should be updated based on the processed media and provide insights into the model's performance.
7. **Error Handling:**
   - The system should handle errors gracefully, providing appropriate messages for unsupported file formats, failed detections, and other issues.
   - The system must ensure that users are informed of any processing errors and provide guidance for resolving them.

2.2 Non-Functional Requirements

Performance:

- The system should process images and videos in a reasonable time frame, aiming for real-time performance.
- The model's inference time should be minimized to ensure a smooth user experience.

Scalability:

- The system should be designed to handle multiple users simultaneously, ensuring that concurrent processing does not degrade performance.
- The system architecture should support scaling, allowing additional computational resources to be added as needed.

Usability:

- The user interface should be intuitive and easy to navigate, ensuring that users with varying technical expertise can use the system effectively.
- Instructions and guidance should be provided to help users understand how to upload or capture media and interpret the results.

Reliability:

- The system should be robust, handling various media formats and conditions without crashing or producing incorrect results.
- The model should be evaluated and validated to ensure reliable emotion detection across different scenarios.

Security:

- The system should implement security measures to protect user data and prevent unauthorized access.
- Uploaded media files should be handled securely, ensuring that user privacy is maintained.

Maintainability:

- The system's codebase should be well-documented, allowing developers to understand and modify the code easily.
- The system architecture should be modular, enabling easy updates and integration of new features.

Compatibility:

- The system should be compatible with different operating systems, including Windows, macOS, and Linux.
- The system should support popular web browsers, ensuring that users can access the application regardless of their preferred browser.

2.3 Safety Requirements

Data Privacy:

- The system must ensure that all uploaded media files are handled securely, with strict access controls to protect user privacy.
- User data, including images and videos, should not be stored or shared without explicit consent.

User Consent:

- The system must obtain user consent before capturing images or videos using the webcam.
- Clear information should be provided to users about how their data will be used and processed.

Error Handling:

- The system should implement robust error handling to prevent crashes and ensure that users receive appropriate feedback for any issues encountered.
- Safety mechanisms should be in place to handle unexpected inputs or conditions, ensuring that the system operates reliably.

Operational Safety:

- The system should not cause any harm or discomfort to users, ensuring that the webcam and other hardware components are used safely.
- The application should include safeguards to prevent overuse or misuse of the webcam, protecting both the hardware and the user's privacy.

2.4 Hardware Requirements

Minimum Hardware Requirements:

- **Processor:** Intel Core i5 or equivalent
- **RAM:** 8GB
- **Storage:** 256GB SSD
- **Graphics:** Integrated GPU (for basic functionality)
- **Webcam:** Built-in or external webcam with a resolution of at least 720p
- **Internet Connection:** Stable internet connection for accessing the web application

Recommended Hardware Requirements:

- **Processor:** Intel Core i7 or equivalent
- **RAM:** 16GB
- **Storage:** 512GB SSD
- **Graphics:** Dedicated GPU (e.g., NVIDIA GeForce GTX 1060 or equivalent) for improved processing speed
- **Webcam:** High-resolution webcam (1080p or higher) for better image and video quality
- **Internet Connection:** High-speed internet connection for seamless operation and faster data processing

Additional Hardware (Optional):

- **External Microphone:** For better audio capture during video recording
- **Lighting:** Proper lighting setup for improved image and video quality during capture

3. Project Design

3.1 Methodology

Overview: The methodology for the Crowd Emotion Recognition project involves several key steps: data collection, preprocessing, model training, deployment, and evaluation. Each step is crucial for developing a robust and effective emotion recognition system.

Steps Involved:

1. **Data Collection:**
   - The primary dataset used for this project is the FER2013 dataset, which contains 35,887 grayscale images of faces categorized into seven emotion classes: angry, disgust, fear, happy, sad, surprise, and neutral.

2. **Data Preprocessing:**
   - **Normalization:** Pixel values are normalized to the range [0, 1] to facilitate faster and more efficient training.
   - **Resizing:** Images are resized to 48x48 pixels to ensure uniform input size for the model.
   - **Data Augmentation:** Techniques such as rotation, zoom, and horizontal flip are applied to increase the diversity of the training data and improve model generalization.
3. **Model Training:**
   - A pre-trained ResNet50 model is fine-tuned on the FER2013 dataset. The model architecture includes several convolutional layers, pooling layers, and fully connected layers to extract features and classify emotions.
   - The model is compiled using the categorical cross-entropy loss function and the Adam optimizer with a learning rate decay schedule to adjust the learning rate during training.
   - Data augmentation is applied during training to prevent overfitting and improve the model's ability to generalize to new data.
4. **Deployment:**
   - The trained model is deployed in a web application built with Streamlit. The application allows users to upload images and videos or capture them using a webcam.
   - The application processes the input media and displays the detected emotions along with visual feedback, such as bounding boxes around detected faces.
5. **Evaluation:**
   - The model's performance is evaluated using metrics such as accuracy, confusion matrix, classification report, and ROC curves.
   - Various data visualizations are provided to help users understand the distribution of emotions and the model's performance.

3.2 Architecture Overview

System Architecture: The Crowd Emotion Recognition system consists of several interconnected components, each responsible for a specific functionality. The key components of the system architecture are:

1. **Frontend:**
   - **User Interface:** Built with Streamlit, the user interface provides an easy-to-use platform for uploading images and videos, capturing media using a webcam, and displaying the results.
   - **Data Visualization Dashboard:** The dashboard displays various data visualizations related to emotion detection and model performance.
2. **Backend:**
   - **Model Server:** Hosts the pre-trained ResNet50 model and handles requests for emotion detection. The model server processes the input media and returns the detected emotions.
   - **Data Processing Module:** Responsible for preprocessing the input media, including resizing, normalization, and face detection.

3. **Database:**
   - **Storage:** Stores the processed media files and metadata, such as detected emotions and timestamps. The database ensures that data can be retrieved and analyzed later.
4. **Integration Layer:**
   - **API Gateway:** Manages communication between the frontend and backend components. The API gateway handles requests from the user interface and forwards them to the appropriate backend services.

Workflow:

1. **Media Upload/Capture:**
   - Users can upload images and videos or capture them using a webcam through the Streamlit interface.
2. **Data Processing:**
   - The uploaded or captured media is preprocessed, including resizing, normalization, and face detection.
3. **Emotion Detection:**
   - The preprocessed media is sent to the model server, which uses the ResNet50 model to detect and classify emotions.
4. **Result Display:**
   - The detected emotions are sent back to the frontend and displayed to the user along with visual feedback.
5. **Data Visualization:**
   - The visualization dashboard provides insights into the distribution of emotions and the model's performance.

3.3 Design Description

Frontend Design:

- **Streamlit Interface:**
  - The user interface is designed using Streamlit, providing a simple and intuitive platform for interaction.
  - Users can upload images and videos or capture them using a webcam.
  - The interface displays the uploaded or captured media along with the detected emotions and visual feedback.
  - A navigation link to the data visualization dashboard is provided for users to view various performance metrics and visualizations.

Backend Design:

- **Model Server:**
  - The model server hosts the pre-trained ResNet50 model and handles requests for emotion detection.
  - The server processes the input media, performs face detection, and classifies emotions.
- **Data Processing Module:**

  - This module is responsible for preprocessing the input media, including resizing, normalization, and face detection.
  - OpenCV is used for face detection, and TensorFlow/Keras is used for image preprocessing and model inference.

Database Design:

- **Storage:**
  - The storage component stores the processed media files and metadata, ensuring that data can be retrieved and analyzed later.
  - The database schema includes tables for storing media files, detected emotions, and timestamps.

Integration Layer:

- **API Gateway:**
  - The API gateway manages communication between the frontend and backend components.
  - It handles requests from the user interface and forwards them to the appropriate backend services for processing.

3.4 Environment

Development Environment:

- **Programming Languages:** Python 3.7+
- **Frameworks and Libraries:**
  - TensorFlow and Keras: For building and training the emotion detection model.
  - OpenCV: For image and video processing.
  - Streamlit: For creating the web application.
  - Seaborn and Matplotlib: For data visualizations.
  - Scikit-learn: For evaluation metrics.
  - Pandas: For data manipulation.

Hardware Environment:

- **Development Machine:**
  - Processor: Intel Core i7 or equivalent
  - RAM: 16GB
  - Storage: 512GB SSD
  - Graphics: Dedicated GPU (e.g., NVIDIA GeForce GTX 1060 or equivalent) for improved processing speed
  - Webcam: High-resolution webcam (1080p or higher)
- **Deployment Server:**
  - Processor: Intel Xeon or equivalent
  - RAM: 32GB
  - Storage: 1TB SSD

  - Graphics: High-end GPU (e.g., NVIDIA Tesla V100) for real-time processing and inference
  - Network: High-speed internet connection for seamless operation and faster data processing

Software Environment:

- **Operating Systems:**
  - Development: Windows 10, macOS, or Linux
  - Deployment: Ubuntu Server or other Linux distributions
- **Tools and IDEs:**
  - Jupyter Notebook: For experimentation and model development
  - Visual Studio Code: For code development and debugging
  - Git: For version control and collaboration
- **Dependencies:**
  - TensorFlow: For deep learning and model training
  - Keras: For building and deploying the ResNet50 model
  - OpenCV: For image and video preprocessing
  - Streamlit: For creating the web application
  - Seaborn and Matplotlib: For data visualization
  - Scikit-learn: For evaluation metrics
  - Pandas: For data manipulation and preprocessing

4. Implementation and Evaluation

4.1 Development Stages

4.1.1 Strategy

The development of the Crowd Emotion Recognition system follows an iterative and incremental strategy. This approach ensures that each component of the system is developed, tested, and integrated progressively, allowing for continuous feedback and improvements.

1. **Requirement Analysis**: Identify and document the system's functional and non-functional requirements.
2. **Design**: Develop the system architecture and detailed design specifications.
3. **Implementation**: Code the system components, including data preprocessing, model training, and the web application.
4. **Testing**: Conduct unit testing, integration testing, and system testing to ensure the system meets the requirements.
5. **Deployment**: Deploy the system on a production server and monitor its performance.
6. **Maintenance**: Continuously monitor and update the system based on user feedback and evolving requirements.

4.1.2 Tools Used

Development Tools:

- **Visual Studio Code**: Integrated Development Environment (IDE) for writing and debugging code.
- **Jupyter Notebook**: For experimenting with data preprocessing, model training, and visualization.

Version Control:

- **Git**: For version control and collaboration.
- **GitHub**: For hosting the project repository and enabling collaboration.

Deployment Tools:

- **Streamlit**: For building and deploying the web application.
- **Docker**: For containerizing the application to ensure consistent deployment across different environments.

Data Visualization:

- **Matplotlib**: For creating static, interactive, and animated visualizations.
- **Seaborn**: For making statistical graphics.

4.1.3 Technologies

Programming Languages:

- **Python**: The primary programming language used for implementing the system.

Libraries and Frameworks:

- **TensorFlow and Keras**: For building and training the deep learning model.
- **OpenCV**: For image and video processing.
- **Streamlit**: For creating the web application.
- **Scikit-learn**: For evaluation metrics and model validation.
- **Pandas**: For data manipulation and preprocessing.

4.1.4 Methodologies

Agile Development:

- The project follows Agile principles, with iterative development cycles and regular feedback loops.
- Daily stand-up meetings and weekly sprints ensure that the development stays on track and any issues are promptly addressed.

Data Augmentation:

- Techniques such as rotation, zoom, and horizontal flip are applied to the training data to improve the model's robustness and generalization capabilities.

Model Training and Validation:

- The model is trained using a combination of training and validation datasets to monitor performance and prevent overfitting.

4.1.5 System Architecture

Frontend:

- **Streamlit Interface**: Provides an intuitive interface for users to upload or capture media, view detected emotions, and access the data visualization dashboard.

Backend:

- **Model Server**: Hosts the pre-trained ResNet50 model and handles emotion detection requests.
- **Data Processing Module**: Preprocesses input media, including resizing, normalization, and face detection.

Database:

- **Storage**: Stores processed media files and metadata for retrieval and analysis.

Integration Layer:

- **API Gateway**: Manages communication between the frontend and backend components.

4.2 System Integrations

Integration with Streamlit:

- The web application built with Streamlit integrates seamlessly with the backend model server and data processing module.
- The Streamlit interface allows users to upload or capture media and displays the detected emotions along with various visualizations.

Integration with OpenCV:

- OpenCV is used for image and video preprocessing, including face detection and resizing.

Integration with TensorFlow/Keras:

- The pre-trained ResNet50 model, built with TensorFlow and Keras, is integrated into the backend for emotion detection.

4.3 User Interface

Streamlit Interface:

- The user interface is designed to be intuitive and user-friendly, allowing users to easily upload or capture media and view the results.
- The interface includes sections for media upload, media capture, result display, and data visualization dashboard.

- Visual feedback, such as bounding boxes around detected faces and emotion labels, is provided to enhance user understanding.

Screenshots:

- Add screenshots of the Streamlit interface, showing the media upload section, captured media, detected emotions, and data visualization dashboard.

4.4 Evaluation

Evaluation Metrics:

- **Accuracy**: The percentage of correct predictions made by the model.
- **Confusion Matrix**: A matrix that visualizes the performance of the model across different emotion categories.
- **Classification Report**: Includes precision, recall, and F1-score for each emotion category.
- **ROC Curves**: Receiver Operating Characteristic curves for each class to evaluate the model's performance.

Data Visualization:

- Various data visualizations are provided to help users understand the distribution of emotions and the model's performance.

4.5 Unit Testing

Unit Testing Approach:

- Individual components of the system, such as data preprocessing functions, model training scripts, and API endpoints, are tested in isolation.
- Unit tests are written to ensure that each component functions as expected and handles edge cases gracefully.

Tools:

- **PyTest**: For writing and executing unit tests.
- **Mock**: For simulating external dependencies and testing components in isolation.

4.6 Functional Testing

Functional Testing Approach:

- Functional tests are conducted to ensure that the system meets the specified requirements and performs as expected.
- Tests include verifying that users can upload or capture media, that the system can detect and classify emotions, and that the results are displayed correctly.

Tools:

- **Selenium**: For automating browser interactions and testing the user interface.
- **Streamlit Testing Tools**: For testing the Streamlit interface and its functionality.

4.7 Testing Requirements

Test Environment:

- The testing environment should replicate the production environment as closely as possible, including the same hardware and software configurations.
- Test data should include a variety of images and videos representing different emotions and conditions.

Test Data:

- The FER2013 dataset is used for testing the model's performance.
- Additional test data, such as images and videos captured from the webcam, are used to evaluate the system's real-time performance.

4.8 Test Cases

Unit Test Cases:

1. **Data Preprocessing**:
   - Verify that images are resized correctly.
   - Ensure that pixel values are normalized to the range [0, 1].
2. **Model Inference**:
   - Check that the model can process input data and return predictions.
   - Validate the accuracy of the model's predictions on sample data.

Functional Test Cases:

1. **Media Upload**:
   - Verify that users can upload images and videos in supported formats.
   - Ensure that the uploaded media is displayed correctly.
2. **Media Capture**:
   - Verify that users can capture images and videos using the webcam.
   - Ensure that the captured media is processed and displayed correctly.
3. **Emotion Detection**:
   - Check that the system can detect and classify emotions from uploaded or captured media.
   - Validate that the detected emotions are displayed with appropriate visual feedback.
4. **Data Visualization**:
   - Ensure that the data visualization dashboard displays the correct plots.
   - Verify that the plots are updated based on the processed media.

Streamlit Interface:

- Add a section with screenshots of the Streamlit interface, showing the various features and functionalities.

4. Implementation and Evaluation

4.1 Development Stages

4.1.1 Strategy

The development of the Crowd Emotion Recognition system follows an iterative and incremental strategy. This approach ensures that each component of the system is developed, tested, and integrated progressively, allowing for continuous feedback and improvements.

1. **Requirement Analysis**: Identify and document the system's functional and non-functional requirements.
2. **Design**: Develop the system architecture and detailed design specifications.
3. **Implementation**: Code the system components, including data preprocessing, model training, and the web application.
4. **Testing**: Conduct unit testing, integration testing, and system testing to ensure the system meets the requirements.
5. **Deployment**: Deploy the system on a production server and monitor its performance.
6. **Maintenance**: Continuously monitor and update the system based on user feedback and evolving requirements.

4.1.2 Tools Used

Development Tools:

- **Visual Studio Code**: Integrated Development Environment (IDE) for writing and debugging code.
- **Jupyter Notebook**: For experimenting with data preprocessing, model training, and visualization.

Version Control:

- **Git**: For version control and collaboration.
- **GitHub**: For hosting the project repository and enabling collaboration.

Deployment Tools:

- **Streamlit**: For building and deploying the web application.
- **Docker**: For containerizing the application to ensure consistent deployment across different environments.

Data Visualization:

- **Matplotlib**: For creating static, interactive, and animated visualizations.
- **Seaborn**: For making statistical graphics.

4.1.3 Technologies

Programming Languages:

- **Python**: The primary programming language used for implementing the system.

Libraries and Frameworks:

- **TensorFlow and Keras**: For building and training the deep learning model.
- **OpenCV**: For image and video processing.
- **Streamlit**: For creating the web application.
- **Scikit-learn**: For evaluation metrics and model validation.
- **Pandas**: For data manipulation and preprocessing.

4.1.4 Methodologies

Agile Development:

- The project follows Agile principles, with iterative development cycles and regular feedback loops.
- Daily stand-up meetings and weekly sprints ensure that the development stays on track and any issues are promptly addressed.

Data Augmentation:

- Techniques such as rotation, zoom, and horizontal flip are applied to the training data to improve the model's robustness and generalization capabilities.

Model Training and Validation:

- The model is trained using a combination of training and validation datasets to monitor performance and prevent overfitting.

4.1.5 System Architecture

Frontend:

- **Streamlit Interface**: Provides an intuitive interface for users to upload or capture media, view detected emotions, and access the data visualization dashboard.

Backend:

- **Model Server**: Hosts the pre-trained ResNet50 model and handles emotion detection requests.
- **Data Processing Module**: Preprocesses input media, including resizing, normalization, and face detection.

Database:

- **Storage**: Stores processed media files and metadata for retrieval and analysis.

Integration Layer:

- **API Gateway**: Manages communication between the frontend and backend components.

4.2 System Integrations

Integration with Streamlit:

- The web application built with Streamlit integrates seamlessly with the backend model server and data processing module.
- The Streamlit interface allows users to upload or capture media and displays the detected emotions along with various visualizations.

Integration with OpenCV:

- OpenCV is used for image and video preprocessing, including face detection and resizing.

Integration with TensorFlow/Keras:

- The pre-trained ResNet50 model, built with TensorFlow and Keras, is integrated into the backend for emotion detection.

4.3 User Interface

Streamlit Interface:

- The user interface is designed to be intuitive and user-friendly, allowing users to easily upload or capture media and view the results.
- The interface includes sections for media upload, media capture, result display, and data visualization dashboard.
- Visual feedback, such as bounding boxes around detected faces and emotion labels, is provided to enhance user understanding.

Screenshots:

- Add screenshots of the Streamlit interface, showing the media upload section, captured media, detected emotions, and data visualization dashboard.

4.4 Evaluation

Evaluation Metrics:

- **Accuracy**: The percentage of correct predictions made by the model.
- **Confusion Matrix**: A matrix that visualizes the performance of the model across different emotion categories.
- **Classification Report**: Includes precision, recall, and F1-score for each emotion category.
- **ROC Curves**: Receiver Operating Characteristic curves for each class to evaluate the model's performance.

Data Visualization:

- Various data visualizations are provided to help users understand the distribution of emotions and the model's performance.

4.5 Unit Testing

Unit Testing Approach:

- Individual components of the system, such as data preprocessing functions, model training scripts, and API endpoints, are tested in isolation.

- Unit tests are written to ensure that each component functions as expected and handles edge cases gracefully.

Tools:

- **PyTest**: For writing and executing unit tests.
- **Mock**: For simulating external dependencies and testing components in isolation.

4.6 Functional Testing

Functional Testing Approach:

- Functional tests are conducted to ensure that the system meets the specified requirements and performs as expected.
- Tests include verifying that users can upload or capture media, that the system can detect and classify emotions, and that the results are displayed correctly.

Tools:

- **Selenium**: For automating browser interactions and testing the user interface.
- **Streamlit Testing Tools**: For testing the Streamlit interface and its functionality.

4.7 Testing Requirements

Test Environment:

- The testing environment should replicate the production environment as closely as possible, including the same hardware and software configurations.
- Test data should include a variety of images and videos representing different emotions and conditions.

Test Data:

- The FER2013 dataset is used for testing the model's performance.
- Additional test data, such as images and videos captured from the webcam, are used to evaluate the system's real-time performance.

4.8 Test Cases

Unit Test Cases:

1. **Data Preprocessing**:
   - Verify that images are resized correctly.
   - Ensure that pixel values are normalized to the range [0, 1].
2. **Model Inference**:
   - Check that the model can process input data and return predictions.
   - Validate the accuracy of the model's predictions on sample data.

Functional Test Cases:

1. **Media Upload**:
   - Verify that users can upload images and videos in supported formats.

   - Ensure that the uploaded media is displayed correctly.
2. **Media Capture**:
   - Verify that users can capture images and videos using the webcam.
   - Ensure that the captured media is processed and displayed correctly.
3. **Emotion Detection**:
   - Check that the system can detect and classify emotions from uploaded or captured media.
   - Validate that the detected emotions are displayed with appropriate visual feedback.
4. **Data Visualization**:
   - Ensure that the data visualization dashboard displays the correct plots.
   - Verify that the plots are updated based on the processed media.

Streamlit Interface:

- Add a section with screenshots of the Streamlit interface, showing the various features and functionalities.