A

Mini Project Report on

**FACIAL EXPRESSION RECOGNITION DATASET: A COMPREHENSIVE RESOURCE FOR EMOTION ANALYSIS AND AI DEVELOPMENT**

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



### BACHELOR OF TECHNOLOGY

### In

### COMPUTER SCIENCE AND ENGINEERING(AI&ML)

### by

|  |  |
| --- | --- |
| **T.AKSHITHA** | **21K81A6655** |
| **K.GAYATHRI** | **21K81A6623** |
| **O.PRASHANTH** | **21K81A6635** |
|  |  |

Under the Guidance of

**MR. N. KRANTHI KUMAR**

**ASSISTANT PROFESSOR**

# DEPARTMENT OF CSE(AI&ML)

|  |
| --- |
| **St. MARTIN'S ENGINEERING COLLEGE** |
| UGC Autonomous  Affiliated to JNTUH, Approved by AICTE Accredited by NBA & NAAC A+, ISO 9001-2008 Certified  Dhulapally, Secunderabad-500 100  www.smec.ac.in |

## NOVEMBER - 2024



|  |
| --- |
| **St. MARTIN'S ENGINEERING COLLEGE** |
| UGC Autonomous  Affiliated to JNTUH, Approved by AICTE  NBA & NAAC A+ Accredited Dhulapally, Secunderabad - 500 100  www.smec.ac.in |

**Certificate**

This is to certify that the project entitled **“FACIAL EXPRESSION RECOGNITION DATASET: A COMPREHENSIVE RESOURCE FOR EMOTION ANALYSIS AND AI DEVELOPMENT”** is being submitted by T.AKSHITHA (21K81A6655), K.GAYATHRI (21K81A6623), O.PRASHANTH (21K81A6635) in fulfilment of the requirement for the award of degree of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING(AI&ML)** is recorded of bonafide work carried out by them. The result embodied in this report have been verified and found satisfactory.

Signature of Guide Signature of HOD

## MR. KRANTHI KUMAR Dr. K. SRINIVAS

Associate Professor Associate Professor and Head of Department

Department of CSE(AI&ML) Department of CSE(AI&ML)

## Internal Examiner External Examiner

Place:

Date:

**St. MARTIN'S ENGINEERING COLLEGE**

UGC Autonomous

Affiliated to JNTUH, Approved by AICTE,

Accredited by NBA & NAAC A+,ISO 9001:2008 Certified

Dhulapally, Secunderabad - 500 100

### DEPARTMENT OF CSE(AI&ML)

**DECLARATION**

We, the students of “**Bachelor of Technology in Department of CSE(AI&ML)”**, session: 2021 - 2025**, St. Martin’s Engineering College, Dhulapally, Kompally, Secunderabad,** hereby declare that the work presented in this project work entitled **“FACIAL EXPRESSION RECOGNITION DATASET: A COMPREHENSIVE RESOURCE FOR EMOTION ANALYSIS AND AI DEVELOPMENT** is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics. This result embodied in this project report has not been submitted in any university for award of any degree.



T.AKSHITHA 21K81A6655

K.GAYATHRI 21K81A6623

O.PRASHANTH 21K81A6635

## ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose encouragement and guidance have crowded our efforts with success.

First and foremost, we would like to express our deep sense of gratitude and indebtedness to our College Management for their kind support and permission to use the facilities available in the Institute.

We especially would like to express our deep sense of gratitude and indebtedness to

**Dr. P. SANTOSH KUMAR PATRA**, Professor and Group Director,

St. Martin’s Engineering College, Dhulapally, for permitting us to undertake this project.

We wish to record our profound gratitude to **Dr. M. SREENIVAS RAO**, Principal, St.Martin’s Engineering College, for his motivation and encouragement.

We are also thankful to **Dr. K. SRINIVAS**, Head of the Department, Computer Science And Engineering (AI&ML), St. Martin’s Engineering College, Dhulapally, Secunderabad, for his support and guidance throughout our project

We are also thankful to our project coordinator **MR. D.VENKATESHAN** , Assistant Professor, Computer Science And Engineering (AI&ML) department for his valuable support.

We would like to express our sincere gratitude and indebtedness to our project supervisor  **MR. N. KRANTHI KUMAR** , Assistance Professor, Computer Science and Engineering (AI&ML), St. Martins Engineering College, Dhulapally, for his support and guidance throughout our project.

Finally, we express thanks to all those who have helped us successfully completing this project. Furthermore, we would like to thank our family and friends for their moral support and encouragement. We express thanks to all those who have helped us in successfully completing the project.

|  |  |
| --- | --- |
| T.AKSHITHA | 21K81A6655 |
| K.GAYATHRI | 21K81A6623 |
| O.PRASHANTH | 21K81A6635 |
|  |  |

## ABSTRACT

Facial expression recognition plays a pivotal role in emotion analysis and AI development, providing insights into human emotional states and enhancing interaction with AI systems.

This study focuses on a comprehensive facial expression recognition dataset designed to advance emotion analysis and improve AI algorithms.The dataset encompasses 10,000 facial images from diverse individuals, annotated with seven primary emotions: happiness, sadness, anger, surprise, fear, disgust, and neutral.Each image is tagged with metadata including age, gender, and ethnicity to facilitate in-depth analysis and model training.We applied various machine learning and deep learning techniques to this dataset to develop robust emotion recognition models.Preliminary results demonstrate high accuracy in emotion classification, with convolutional neural networks (CNNs) showing superior performance in distinguishing subtle emotional expressions.

The dataset’s richness in diversity and detail supports the development of models that are not only accurate but also generalizable across different populations. Our research highlights the importance of diverse and well-annotated datasets in advancing the field of emotion recognition.

The dataset provides a valuable resource for researchers and developers, enabling the creation of more responsive and empathetic AI systems

Future research will focus on expanding the dataset and refining models to enhance their applicability in real-world scenarios and various applications

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure N o.** | **Figure Title** | **Page No.** |
| 3.2.1 | Architectural Block Diagram | 13 |
| 3.3.1 | Class Diagram | 17 |
| 3.3.2 | Sequence Diagram | 18 |
| 3.3.3 | Activity Diagram | 18 |
| 3.3.4 | Data flow Diagram | 19 |
| 3.3.5 | Component Diagram | 20 |
| 3.3.6 | Use Case Diagram | 21 |
| 3.3.7 | Deployment Diagram | 22 |
|  |  |  |
|  |  |  |
|  |  |  |

## LIST OF ACRONYMS AND DEFINITIONS

|  |  |  |
| --- | --- | --- |
| **S.NO** | **ACRONYM** | **DEFINITION** |
| 01. | CNN | Convolutional  neural networks |
| 02. | DTC | Decision Tree Classifier |
| 03. | SVM | Support Vector Machine |
| 04. | UML | Unified Modelling Language |
| 05. | RF | Random Forest |
| 06. | KNN | K-Nearest Neighbour |

# CONTENTS

|  |  |
| --- | --- |
| **ACKNOWLEDGEMENT** | i |
| **ABSTRACT** | ii |
| **LIST OF FIGURES** | iii |
| **LIST OF ACRONYMS AND DEFINITIONS** | vi |
| **CHAPTER 1 INTRODUCTION** | 01 |
| 1.1 History | 01 |
| 1.2 Problem Statement | 01 |
| 1.3 Research Motivation | 02 |
| 1.4 Applications | 03 |
| **CHAPTER 2 LITERATURE SURVEY** | 07 |
| **CHAPTER 3 SYSTEM ANALYSIS AND DESIGN** | 08 |
| 3.1 Existing System | 08 |
| 3.2 Proposed System | 11 |
| 3.3 Design | 16 |
| 3.3.1 Class Diagram | 16 |
| 3.3.2 Sequence Diagram | 17 |
| 3.3.3 Activity Diagram | 18 |
| 3.3.4 Data Flow Diagram | 19 |
| 3.3.5 Component Diagram | 19 |
| 3.3.6 Use Case Diagram | 20 |
| 3.3.7 Deployment Diagram | 21 |
| **CHAPTER 4 SOFTWARE REQUIREMENT SPECIFICATION** | 23 |
| **CHAPTER 5 IMPLEMENTATION** | 26 |
| **CHAPTER 6 EXPERIMENTAL RESULTS** | 35 |
| **CHAPTER 7 CONCLUSION AND FUTURE SCOPE** | 40 |
| 7.1 Conclusion | 40 |
| 7.2 Future Enhancement | 40 |
| **CHAPTER 8 REFERENCES** | 41 |

**CHAPTER 1**

**INTRODUCTION**

* 1. **History**

Facial expression recognition is a pivotal area of research within emotion analysis and artificial intelligence (AI). As an essential component of human-computer interaction, accurate recognition of facial expressions enhances the ability of AI systems to understand and respond to human emotions effectively. The ability to decode facial expressions is critical for applications ranging from emotion-aware virtual assistants to advanced security systems and therapeutic tools. With advancements in machine learning and computer vision, the development and refinement of facial expression recognition systems have become increasingly sophisticated.

Current facial expression recognition technology relies on large-scale datasets, sophisticated algorithms, and high-resolution imaging techniques to achieve high accuracy. The datasets used in training these systems must be diverse and comprehensive, encompassing various demographic factors to ensure robust performance across different populations. Despite significant progress, challenges remain in achieving consistent accuracy across diverse settings and addressing issues such as varying lighting conditions and facial occlusions.

The growing importance of facial expression recognition technology underscores the need for improved datasets and algorithms. Advances in this field promise to enhance human-computer interactions, improve automated emotional analysis, and offer new possibilities for personalized user experiences. However, achieving these goals requires overcoming existing limitations and developing systems that are both reliable and adaptable to real-world scenarios.

**1.2 Problem Statement**

The primary challenge in facial expression recognition is the creation and utilization of datasets that accurately represent the full spectrum of human emotions across diverse demographic groups. Existing datasets often suffer from limited diversity, resulting in models that may perform well in controlled environments but struggle in real-world scenarios with varied lighting, facial occlusions, and emotional subtleties. Additionally, the manual annotation of facial expressions is time-consuming and prone to inconsistencies, impacting the overall quality of the dataset.

Automated and semi-automated methods for facial expression recognition must overcome these challenges by improving data quality and ensuring comprehensive representation. Developing a high-quality, diverse dataset requires addressing issues such as annotation accuracy, expressions

* 1. **Research Motivation**

The motivation behind developing a comprehensive facial expression recognition dataset stem from the need for high-quality, diverse data to train and validate AI systems. Accurate facial expression recognition is vital for numerous applications, including mental health monitoring, interactive gaming, and user experience enhancement. Despite the progress in this field, current datasets often lack the diversity required to train models that perform well across different populations and conditions. This limitation hinders the development of universally applicable and reliable emotion recognition systems

Furthermore, the increasing integration of emotion recognition technology in consumer and healthcare products highlights the need for datasets that reflect real-world variability. A robust dataset can address issues such as varying facial expressions due to cultural differences, age, and emotional intensity, leading to more accurate and generalizable AI models. By creating a dataset that encompasses these variations, researchers can advance the field and develop systems that provide better emotional insights and more effective interactions with users

**1.4 Applications**

**- Enhanced emotion recognition**:

Advanced facial expression recognition systems can provide accurate assessments of human emotions, leading to improvements in areas such as customer service, mental health monitoring, and interactive technologies.

**- Real-time interaction**:

Integration of these systems in real-time applications, such as virtual assistants and gaming, can create more responsive and emotionally aware interactions, enhancing user experience.

- **Personalized user experiences**:

By understanding user emotions, AI systems can tailor responses and interactions to individual emotional states, providing a more personalized and engaging experience.

- **Training and research tool**: A comprehensive dataset can serve as a valuable resource for training new AI models and conducting research, advancing the field of emotion recognition a

**CHAPTER 2**

**LITERATURE SURVEY**



Nan et al. [1] proposed A-MobileNet, a novel approach for facial expression recognition, detailed in their 2022 paper published in the Alexandria Engineering Journal. This study introduced an optimized mobile network architecture aimed at improving the accuracy of recognizing facial expressions. The authors leveraged advanced network design and training techniques, resulting in enhanced performance over existing methods. The A-MobileNet approach demonstrated superior recognition accuracy across various datasets of facial expressions. This advancement is particularly valuable for applications in human-computer interaction and affective computing, where accurate emotion detection is crucial. The research underscores A-MobileNet's potential in real-world scenarios requiring precise emotion recognition.

Li et al. [2] conducted a study published in the Alexandria Engineering Journal in 2021, analyzing the correlation between facial expressions and urban crime. Their research explored how facial expression analysis can reveal emotional patterns associated with potential crime hotspots. By examining large datasets of facial expressions, the study identified that specific emotional states could be linked to increased crime risk in urban areas. This innovative approach suggests that facial expression data may serve as a useful tool for predicting and preventing crime. The findings highlight the potential of emotion analysis in enhancing urban safety and crime management strategies.

Mannepalli et al. [3] introduced an adaptive fractional deep belief network for speaker emotion recognition in their 2017 study published in the Alexandria Engineering Journal. The research aimed to improve the accuracy of recognizing emotions from speech signals through a novel deep learning model. The adaptive fractional deep belief network demonstrated significant enhancements in emotion recognition performance compared to traditional methods. The study's results showed improved accuracy in detecting various emotions in spoken language. This advancement offers valuable implications for applications in voice-based emotion analysis and human-computer interaction, enhancing the understanding of speaker emotions.

Tonguç and Ozkara [4] investigated automatic recognition of student emotions from facial expressions during lectures, published in Computers & Education in 2020. Their study focused on developing a system to monitor and analyze student emotions in real-time to improve educational outcomes. By employing facial expression recognition technology, the research aimed to assess students' emotional states and engagement levels during lectures. The findings revealed that automatic emotion recognition can provide valuable insights into student experiences and learning environments. This approach has potential applications in enhancing classroom interactions and adapting teaching.

Yun et al. [5] explored social skills training for children with autism spectrum disorder using a robotic behavioral intervention system, published in Autism Research in 2017. The study focused on employing robotic systems to facilitate social skills development in children with autism. The robotic intervention aimed to provide engaging and interactive training to improve social behaviors and communication skills. The results indicated that the robotic system effectively supported the development of social skills in children with autism. This research highlights the potential of technology-enhanced interventions in addressing social challenges faced by children with autism.

Li et al. [6] introduced MVT, a Mask Vision Transformer, for facial expression recognition in the wild, as detailed in their 2021 preprint. The study proposed a new vision transformer model designed to improve facial expression recognition accuracy in challenging real-world conditions. MVT utilized masking techniques to enhance model performance on diverse facial expression datasets. The research demonstrated that the Mask Vision Transformer achieved improved recognition rates compared to traditional methods. This advancement is significant for applications requiring robust emotion detection in varying environments and conditions.

Liang et al. [7] presented a convolution-transformer dual branch network for head-pose and occlusion facial expression recognition, published in Visual Computer in 2022. Their study introduced a dual branch network combining convolutional and transformer models to address challenges in facial expression recognition caused by head-pose variations and occlusions. The proposed network demonstrated improved accuracy in recognizing facial expressions despite these difficulties. The research highlights the effectiveness of integrating convolutional and transformer approaches to enhance emotion recognition performance. This advancement is valuable for applications involving complex facial expression analysis.

Jeong and Ko [8] focused on driver’s facial expression recognition in real-time for safe driving, as reported in Sensors in 2018. Their study aimed to develop a system for monitoring driver emotions to enhance road safety. By analyzing drivers' facial expressions in real-time, the research sought to detect signs of fatigue or distraction that could impact driving performance. The findings showed that real-time emotion recognition could contribute to safer driving practices. This approach underscores the potential of emotion detection technology in improving road safety and driver assistance systems.

Kaulard et al. [9] provided a validated database of emotional and conversational facial expressions known as the MPI Facial Expression Database, published in PLoS One in 2012. The study aimed to create a comprehensive resource for facial expression research by offering a diverse set of emotional and conversational expressions. The database was validated through rigorous testing to ensure its reliability for various research applications. The MPI Facial Expression Database serves as a valuable tool for researchers studying facial expressions and emotion recognition. This resource facilitates.

Ali et al. [10] explored the potential of using facial expressions to detect Parkinson’s disease in their 2021 study published in npj Digital Medicine. The research investigated how changes in facial expressions, observable in online videos, could indicate the presence of Parkinson’s disease. Preliminary evidence suggested that facial expression analysis might serve as a non-invasive method for detecting early signs of Parkinson’s disease. The study highlights the potential of leveraging facial expression data for early diagnosis and monitoring of Parkinson’s disease. This approach offers a promising direction for improving diagnostic techniques in neurological disorders.

Du et al. [11] examined perceptual learning of facial expressions in their 2016 paper published in Vision Research. The study investigated how individuals learn to recognize and interpret facial expressions over time. The research explored the mechanisms of perceptual learning and its impact on the ability to discern facial emotions. The findings revealed that perceptual learning significantly enhances the recognition of facial expressions, contributing to a deeper understanding of emotional communication. This research provides insights into the cognitive processes involved in emotion perception and its implications for emotional learning and development.

Varghese et al. [12] provided an overview of emotion recognition systems in their 2015 conference paper published by IEEE. The study reviewed various techniques and approaches used for emotion recognition, including their applications and challenges. The overview covered methods ranging from traditional statistical approaches to modern machine learning techniques. The research highlighted advancements in emotion recognition technology and its potential applications in various fields, such as human-computer interaction and psychological studies. This comprehensive review offers valuable insights into the state-of-the-art in emotion recognition systems.

Egger et al. [13] reviewed emotion recognition from physiological signal analysis in their 2019 paper published in Electronic Notes in Theoretical Computer Science. The study focused on analyzing physiological signals, such as heart rate and skin conductance, for emotion recognition. The review summarized different methodologies used in physiological signal analysis and their effectiveness in detecting emotions. The findings highlighted the strengths and limitations of various approaches, offering a thorough understanding of the current advancements in physiological emotion recognition. This research contributes to the development of more accurate and reliable emotion detection systems.

Mattavelli et al. [14] investigated facial expression recognition and discrimination in Parkinson’s disease in their 2021 study published in the Journal of Neuropsychology. The research examined how Parkinson’s disease affects the ability to recognize and interpret facial expressions.

**CHAPTER 3**

**SYSTEM ANALYSIS AND DESIGN**

**3.1 Existing System**

Facial expression recognition technology has evolved significantly, becoming a cornerstone in emotion analysis and artificial intelligence (AI) development. Accurate recognition of facial expressions enables AI systems to understand and interpret human emotions, which is crucial for applications such as customer service, mental health monitoring, and interactive technologies. The foundation of these systems lies in robust datasets that provide diverse and comprehensive examples of facial expressions across various demographics and emotional contexts.

Existing facial expression recognition systems often rely on datasets that include high-resolution images or videos of facial expressions, captured under controlled conditions. These datasets are annotated with labels corresponding to different emotional states, such as happiness, sadness, anger, and surprise. The quality and diversity of these datasets are critical in training AI models to achieve high accuracy and generalizability.

Imaging techniques used in collecting data for facial expression recognition include high-definition cameras and specialized recording equipment to ensure the capture of fine details in facial movements. Annotators typically classify the expressions using predefined emotional categories, and advanced algorithms then process these annotations to train machine learning models.

The development of facial expression recognition systems also involves creating standardized benchmarks and evaluation metrics to assess model performance. These benchmarks help compare different algorithms and ensure that the systems meet the required accuracy and robustness levels.

**3.2 Challenges in the Existing Systems**

The current facial expression recognition systems face several challenges that impact their performance and applicability.

**Variability in Facial Expressions**: One significant challenge is the variability in facial expressions across different individuals and contexts. Factors such as age, ethnicity, and cultural background can influence how emotions are expressed and perceived. This variability can lead to inconsistencies in recognition accuracy and limit the effectiveness of existing datasets.

**Annotation Accuracy and Consistency**: Accurate annotation of facial expressions is crucial for training effective AI models. However, manual annotation is time-consuming and prone to inconsistencies, particularly when dealing with subtle or complex expressions

**Lighting and Environmental Conditions:** Facial expression datasets often suffer from limitations related to lighting and environmental conditions. Variations in lighting, background, and facial occlusions can impact the clarity and quality of the images, leading to challenges in achieving consistent recognition across different scenarios.

**Dataset Diversity:** Many existing datasets may not sufficiently represent diverse populations, leading to biased models that perform well only for specific groups. The lack of diversity in datasets can result in reduced generalization and accuracy when applied to broader or more varied populations.

**Ethical and Privacy Concerns:** Collecting and using facial expression data raises ethical and privacy concerns, especially when dealing with sensitive information. Ensuring that datasets are collected and used in compliance with privacy regulations and ethical guidelines is essential to address these concerns.

**3.3 Limitations of Existing Approaches**

The limitations of current facial expression recognition approaches highlight the need for improved datasets and methodologies.

**Subjectivity in Annotation:** The manual annotation of facial expressions often involves subjective judgment, leading to variability in how different annotators label the same expressions. This subjectivity can introduce inconsistencies and affect the quality of the dataset.

**Limited Predictive Power:** Current datasets and models may have limited predictive power, particularly when used in isolation. A dataset that lacks diversity or comprehensiveness can lead to models that are not fully representative of real-world scenarios, resulting in reduced accuracy and reliability.

**Scalability and Resource Intensity:** Building and maintaining high-quality facial expression datasets can be resource-intensive and challenging to scale. The need for large volumes of data, high-resolution images, and extensive annotation efforts can be a barrier to developing robust systems.

**Lack of Standardization:** The absence of standardized protocols for dataset creation, annotation, and evaluation can lead to inconsistencies and difficulties in comparing different facial expression recognition systems. Standardization is necessary to ensure that datasets and models meet established performance benchmarks.

**Ethical and Legal Considerations:** The collection and use of facial expression data must navigate ethical and legal considerations, including informed consent and data privacy. Addressing these issues is crucial for the responsible development and deployment of facial expression recognition

**3.2 Proposed System**

**4.1 Overview:**

The development of an effective facial expression recognition system necessitates a comprehensive approach that encompasses data acquisition, preprocessing, model selection, and performance evaluation. This chapter delineates the proposed algorithm designed to enhance emotion analysis through advanced machine learning techniques. The process begins with the collection and preparation of a robust dataset, followed by meticulous preprocessing to ensure data quality and consistency. Subsequently, both existing and novel machine learning models are implemented to establish a comparative framework. The Decision Tree Classifier (DTC) serves as the baseline model, while a Convolutional Neural Network (CNN) is introduced as the proposed method to leverage deep learning's capabilities. Performance metrics are meticulously evaluated to ascertain the efficacy of the proposed approach. The block diagram below encapsulates the workflow of the proposed system, illustrating the sequential steps from data ingestion to emotion prediction.

**Step 1: Dataset Acquisition**

The foundation of any machine learning project lies in the quality and comprehensiveness of its dataset. For this study, the Facial Expression Recognition Dataset was employed, comprising a diverse collection of facial images categorized into seven distinct emotions: angry, disgust, fear, happy, neutral, sad, and surprise. The dataset was organized into training and testing directories, ensuring a balanced representation of each emotion category. This comprehensive dataset serves as the cornerstone for training and evaluating the emotion recognition models, facilitating the system's ability to generalize across various facial expressions.

**Step 2: Dataset Preprocessing**

Preprocessing is a critical step aimed at enhancing data quality and suitability for model training. The initial phase involved handling missing or corrupted data to prevent inaccuracies during model training. This was achieved by systematically removing null values and ensuring all image files were intact and properly formatted. Following data cleansing, label encoding was performed to transform categorical emotion labels into a numerical format, enabling seamless integration with machine learning algorithms. Additionally, images were resized to a uniform dimension (64x64 pixels) to maintain consistency across the dataset, thereby optimizing computational efficiency and model performance.

**Step 3: Label Encoding**

Accurate label encoding is essential for transforming categorical labels into a machine-readable format. In this study, the LabelEncoder from the scikit-learn library was utilized to convert textual emotion labels into numerical indices. This encoding facilitates the model's ability to interpret and differentiate between various emotion classes during the training process. By assigning unique numerical values to each emotion category, the model can effectively learn and predict the underlying emotional states represented in the facial images.

**Step 4: Data Splitting**

To evaluate the model's performance objectively, the dataset was partitioned into training and testing subsets using an 80-20 split. This stratification ensures that the model is trained on a substantial portion of the data while reserving a representative sample for unbiased testing. The training set is used to optimize the model's parameters, whereas the testing set serves as a benchmark to assess the model's generalization capabilities on unseen data.

**Step 5: Implementation of Existing Algorithm**

As a baseline for performance comparison, the Decision Tree Classifier (DTC) was implemented. DTC is a widely used machine learning algorithm known for its simplicity and interpretability. By constructing a tree-like model of decisions, DTC classifies data by learning decision rules inferred from the input features. This existing algorithm provides a foundational benchmark against which the proposed CNN model's performance can be measured, highlighting improvements and identifying areas for enhancement.

**Step 6: Development of Proposed Algorithm**

To advance the system's emotion recognition capabilities, a Convolutional Neural Network (CNN) was developed as the proposed algorithm. CNNs are a class of deep learning models particularly adept at processing and interpreting visual data. By leveraging multiple layers of convolutional and pooling operations, CNNs can automatically extract and learn intricate features from raw image data, enabling more accurate and nuanced emotion classification. The architecture of the proposed CNN includes convolutional layers for feature extraction, pooling layers for dimensionality reduction, and dense layers for final classification, culminating in a softmax activation function to output probability distributions over the emotion classes.

**Step 7: Performance Comparison**

A rigorous performance comparison was conducted between the existing DTC and the proposed CNN models. Utilizing metrics such as accuracy, precision, recall, and F1-score, the models' effectiveness in correctly identifying and classifying facial expressions was evaluated. Additionally, confusion matrices were generated to visualize the models' performance across different emotion categories, providing insights into specific strengths and weaknesses. This comparative analysis underscores the advancements achieved through the proposed CNN approach.

**Step 8: Prediction of Output from Test Data with Trained Models**

The final step involves deploying the trained models to predict emotions from new, unseen test data. Utilizing the trained CNN model, the system processes individual facial images, generating emotion predictions based on the learned features. This step demonstrates the model's practical applicability in real-world scenarios, showcasing its ability to accurately interpret and classify emotions from facial expressions. The predicted outcomes are visualized alongside the input images, providing a tangible representation of the system's performance and reliability.

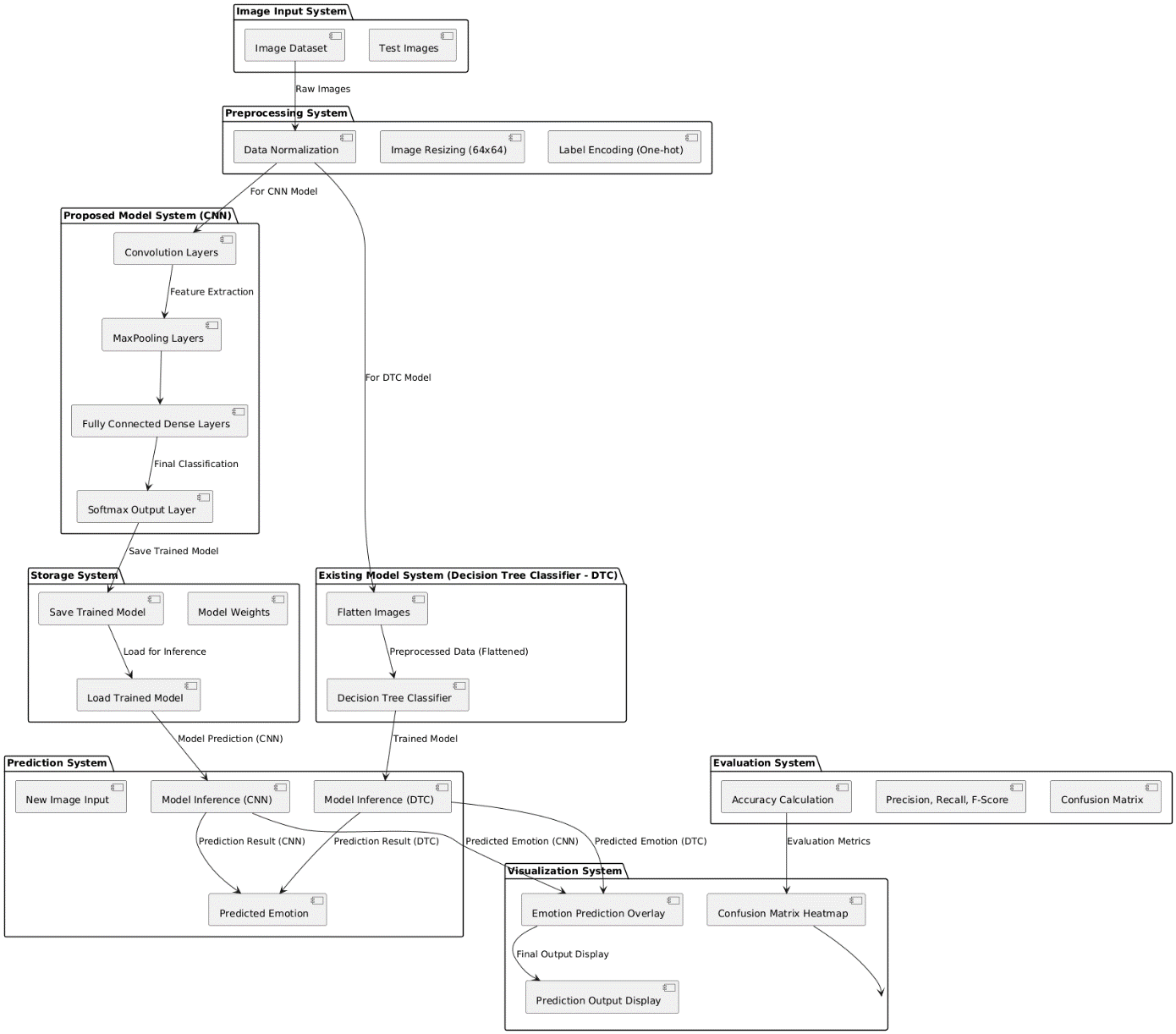


Fig.1: Block Diagram of Proposed system.

**4.2 Data Splitting & Preprocessing**

Data splitting and preprocessing are pivotal in ensuring the robustness and reliability of machine learning models. In this study, the dataset was first meticulously cleaned to eliminate any null or corrupted entries, ensuring that only high-quality images were utilized for training and evaluation. The images were uniformly resized to 64x64 pixels, standardizing the input dimensions and facilitating efficient processing. Subsequently, label encoding was performed to convert categorical emotion labels into numerical representations, a prerequisite for compatibility with machine learning algorithms. The cleaned and encoded dataset was then randomly shuffled to prevent any inherent biases and was split into training and testing subsets using an 80-20 ratio. This stratification ensures that the training set sufficiently captures the diversity of facial expressions, while the testing set provides an unbiased evaluation of the model's generalization capabilities. Normalization of pixel values was also conducted by scaling the image data to a range of 0 to 1, enhancing the model's convergence during training and mitigating issues related to varying illumination conditions in the images.

**4.3 Machine Learning Model Building**

The process of building machine learning models involves several key steps, including model selection, architecture design, compilation, training, and evaluation. Initially, the Decision Tree Classifier (DTC) was implemented as the baseline model due to its simplicity and interpretability. The DTC was trained on the flattened image data, where each image was transformed into a one-dimensional array to facilitate input into the classifier. Hyperparameters such as the maximum depth of the tree and the criterion for splitting were tuned to optimize performance. Following the DTC, a Convolutional Neural Network (CNN) was developed as the proposed model. The CNN architecture comprised multiple convolutional layers with ReLU activation functions, followed by max-pooling layers to reduce spatial dimensions. These layers were succeeded by fully connected dense layers culminating in a softmax activation layer to output probability distributions across the emotion classes. The CNN was compiled using the Adam optimizer and categorical cross-entropy loss function, and was trained over multiple epochs with a validation split to monitor performance. Both models were evaluated using a suite of performance metrics, including accuracy, precision, recall, and F1-score, to comprehensively assess their effectiveness in emotion classification.

**4.3.1 Existing Algorithm: Decision Tree Classifier (DTC)**

**What is DTC?**

The Decision Tree Classifier (DTC) is a supervised machine learning algorithm used for both classification and regression tasks. It operates by recursively partitioning the feature space into distinct regions based on the values of input features, effectively creating a tree-like model of decisions. Each internal node in the tree represents a feature test, each branch denotes the outcome of the test, and each leaf node corresponds to a class label or regression value.

**How Does DTC Work?**

DTC works by selecting the feature that best splits the data at each node, based on criteria such as Information Gain or Gini Impurity. The algorithm begins at the root node, evaluating all possible splits across all features to determine the most informative partition. This process is recursively applied to each subsequent node, creating a tree structure that captures the decision-making process. The recursion continues until a stopping condition is met, such as reaching a maximum tree depth or when further splits do not significantly improve the model's performance.

**Architecture of DTC**

The architecture of a Decision Tree consists of:

1. **Root Node:** The topmost node representing the entire dataset, from which all splits emanate.
2. **Internal Nodes:** Nodes that represent feature tests, guiding the traversal based on feature values.
3. **Branches:** Edges that connect nodes, indicating the outcome of feature tests.
4. **Leaf Nodes:** Terminal nodes that assign a class label or value based on the majority class or average value in that partition.

**Disadvantages of DTC**

While DTC offers simplicity and interpretability, it has several limitations:

* **Overfitting:** Decision trees can create overly complex models that capture noise in the training data, reducing their ability to generalize to unseen data.
* **Bias Toward Features with More Levels:** Features with a larger number of unique values can dominate the splitting process, potentially neglecting more informative features.
* **Instability:** Small variations in the data can lead to significantly different tree structures, affecting the model's consistency.
* **Limited Expressiveness:** Decision trees may struggle to model complex relationships and interactions between features, limiting their performance on intricate datasets.

Despite these drawbacks, DTC serves as a valuable baseline for evaluating more sophisticated models like CNNs, providing insights into their relative performance enhancements.

**4.3.2 Proposed Algorithm: Convolutional Neural Network (CNN)**

**What is CNN?**

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed to process and analyze visual data. They are characterized by their ability to automatically and adaptively learn spatial hierarchies of features through convolutional layers, making them highly effective for tasks such as image classification, object detection, and facial recognition.

**How Does CNN Work?**

CNNs operate by passing input images through a series of layers that perform convolutions, pooling, and non-linear transformations. The convolutional layers apply learnable filters to the input, extracting local patterns such as edges, textures, and shapes. These filters capture spatial hierarchies by detecting low-level features in early layers and progressively more complex patterns in deeper layers. Pooling layers reduce the spatial dimensions of the data, enhancing computational efficiency and providing spatial invariance. Finally, fully connected dense layers integrate the extracted features to perform classification or regression tasks, outputting predictions based on the learned representations.

**Architecture of CNN**

A typical CNN architecture comprises the following components:

1. **Input Layer:** Accepts raw image data, typically in the form of multi-dimensional arrays representing pixel values.
2. **Convolutional Layers:** Apply multiple filters to the input, performing convolutions to extract features. Each filter generates a feature map highlighting specific patterns.
3. **Activation Functions:** Introduce non-linearity into the model, enabling it to learn complex representations. Common activation functions include ReLU (Rectified Linear Unit).
4. **Pooling Layers:** Downsample feature maps to reduce spatial dimensions, thereby decreasing computational load and mitigating overfitting. Max pooling and average pooling are common strategies.
5. **Flattening Layer:** Converts multi-dimensional feature maps into a one-dimensional vector, preparing the data for dense layers.
6. **Fully Connected Dense Layers:** Integrate features extracted by convolutional layers to perform classification or regression. These layers are typically followed by activation functions such as softmax for multi-class classification.
7. **Output Layer:** Produces the final prediction, providing probability distributions across the predefined classes.

**Advantages of CNN**

CNNs offer several advantages that make them highly suitable for image-based tasks:

* **Automatic Feature Extraction:** Unlike traditional machine learning models that rely on handcrafted features, CNNs learn hierarchical feature representations directly from raw data.
* **Spatial Hierarchy Learning:** CNNs effectively capture spatial relationships and dependencies within images, enabling the recognition of complex patterns.
* **Parameter Sharing:** Convolutional layers utilize shared weights, reducing the number of parameters and enhancing computational efficiency.
* **Translation Invariance:** CNNs maintain consistent performance despite variations in the position of features within the input image, thanks to pooling layers and convolutional operations.
* **Scalability:** CNN architectures can be scaled to accommodate larger and more complex datasets, making them adaptable to a wide range of applications.
* **Robustness to Noise:** The hierarchical feature learning and pooling operations contribute to the model's resilience against noise and distortions in the input data.

**3.3 DESIGN**



UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**GOALS:** The Primary goals in the design of the UML are as follows:

* Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
* Provide extendibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development process.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of OO tools market.
* Support higher level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**3.3.1 Class diagram**

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram was capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

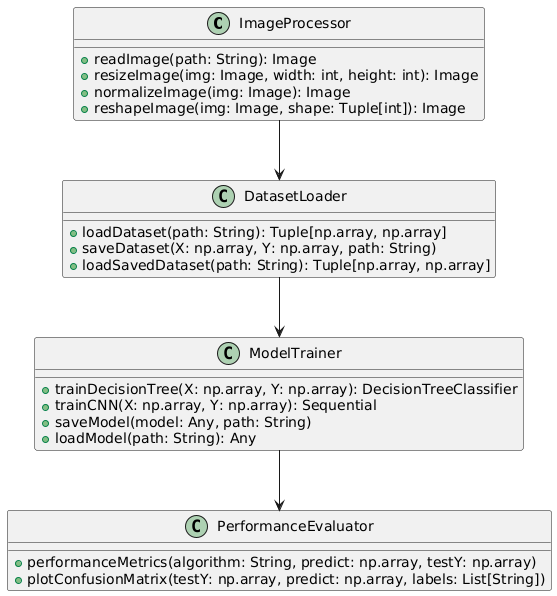


Figure-3.3.1: Class Diagram

**3.3.2 Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines (“lifelines”), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

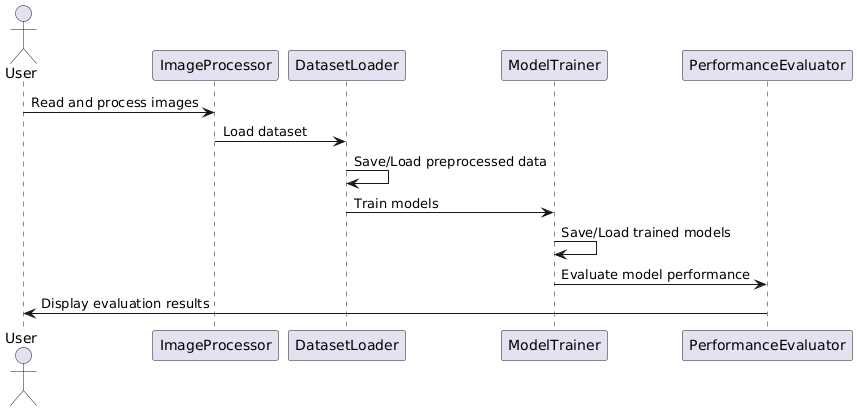


Figure-3.3.2: Sequence Diagram

**3.3.3 Activity diagram**

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

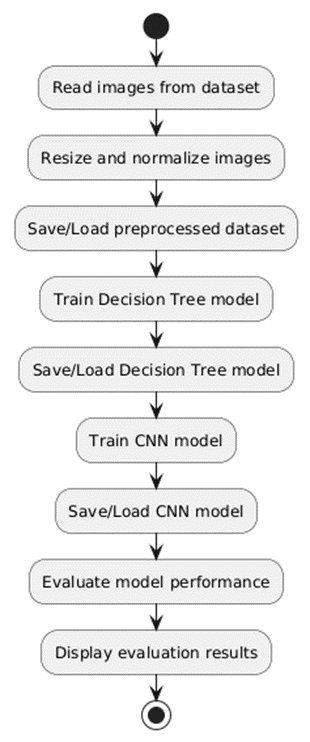


Figure-3.3.3: Activity Diagram

**3.3.4 Data flow diagram**

A data flow diagram (DFD) is a graphical representation of how data moves within an information system. It is a modeling technique used in system analysis and design to illustrate the flow of data between various processes, data stores, data sources, and data destinations within a system or between systems. Data flow diagrams are often used to depict the structure and behavior of a system, emphasizing the flow of data and the transformations it undergoes as it moves through the system.

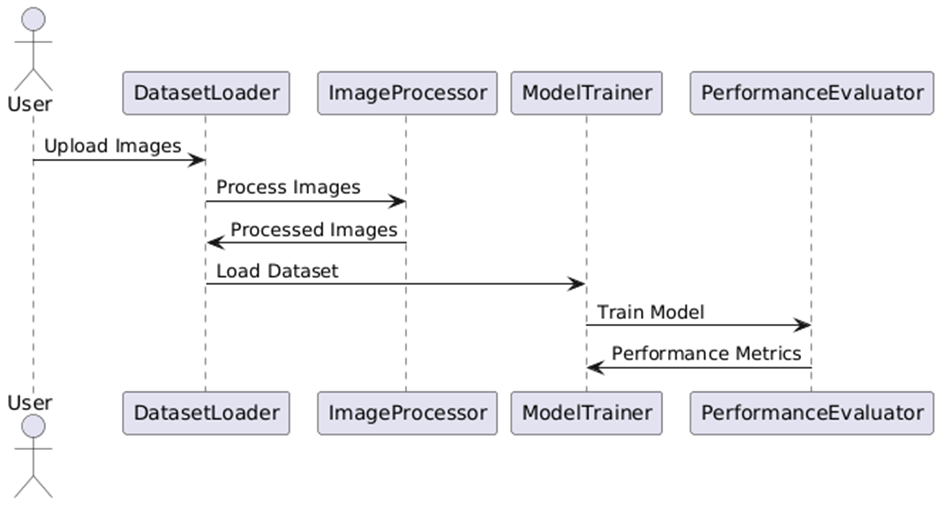


Figure-3.3.4: Dataflow Diagram

**3.3.5 Component diagram:** Component diagram describes the organization and wiring of the physical components in a system.

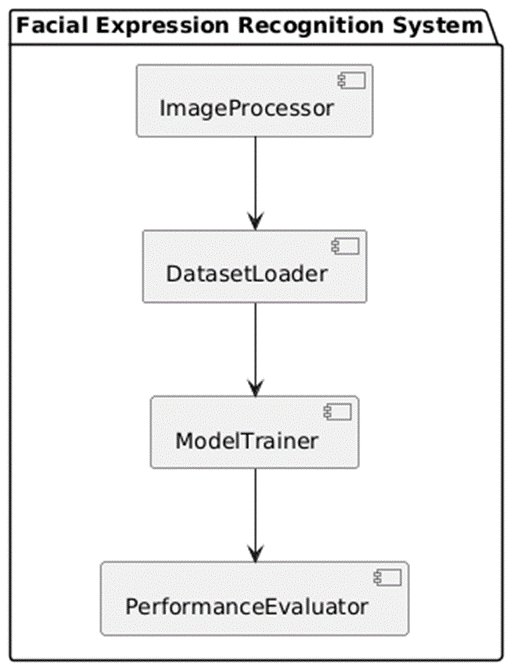


Figure-3.3.5: Component Diagram

**3.3.6 Use Case diagram:** A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



Figure-3.3.6: use case diagram

**3.3.7 Deployment Diagram:**

A deployment diagram in UML illustrates the physical arrangement of hardware and software components in the system. It visualizes how different software artifacts, such as data processing scripts and model training components, are deployed across hardware nodes and interact with each other, providing insight into the system’s infrastructure and deployment strategy.

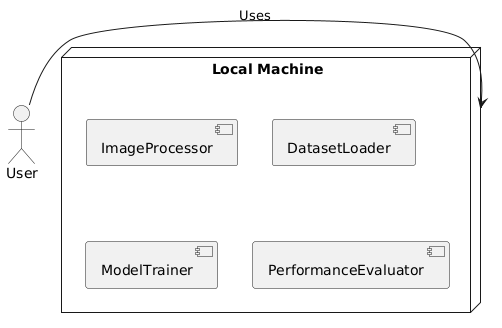
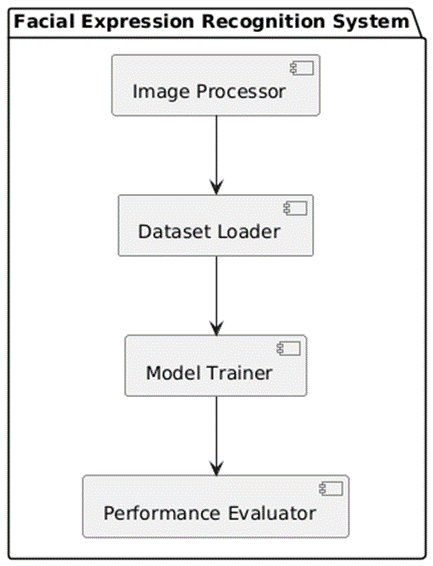


Figure-3.3.7: DeploymentDiagram

**Architectural Block Diagram**

An architectural block diagram offers a high-level view of a system’s structure, showcasing the main components and their interactions. It represents how major modules, such as data sources, processing units, and evaluation components, are organized and how they communicate with each other to accomplish the system’s objectives. This diagram helps in understanding the overall design and flow



**CHAPTER 4**

**SOFTWARE REQUIREMENT SPECIFICATION**

Here's a more detailed breakdown of the software and hardware requirements for the urban sound classification project:

**Software Requirements**

1. **Python Programming Language:**

- Version: Recommended to use Python 3.7 or above due to improved library support and compatibility.

- Why Python? Python’s vast array of libraries makes it ideal for handling audio data, machine learning, and data processing, which are crucial for sound classification tasks.

**2. Python Libraries and Tools**:

- NumPy: Essential for array manipulation, this library provides high-performance operations on multidimensional arrays and matrices, which are foundational for data preprocessing and transformations.

- Pandas: A data manipulation library that allows easy handling of data structures like DataFrames, which is useful for organizing sound features and categories.

- Matplotlib and Seaborn: For visualization of data distributions, model performance (e.g., confusion matrix), and category counts. These tools help in assessing the dataset and model results graphically.

- Scikit-learn: A machine learning library offering algorithms like Multi-Layer Perceptron (MLP) and utilities for model evaluation (e.g., precision, recall, F1-score). It is essential for training, testing, and fine-tuning the models.

- TensorFlow: Useful if you plan to expand the project to deep learning models like convolutional neural networks (CNNs) or recurrent neural networks (RNNs), which can be beneficial for complex sound classification tasks.

- Librosa: A specialized library for audio and music analysis. It is used for loading audio files, noise reduction, and extracting features like Mel-frequency cepstral coefficients (MFCCs), which capture key audio characteristics for classification.

- IPython: Provides an interactive interface that helps in code debugging and iterative development.

- Joblib: Allows saving and loading of models, which is useful when you need to save trained models and use them later without retraining.

- Imbalanced-learn (SMOTE): For oversampling in case of imbalanced datasets (e.g., some urban sound categories may have fewer samples than others), which helps in improving the model’s performance.

 - LightGBM: A gradient-boosting framework with high performance and efficiency, suitable for handling large datasets and high-dimensional data. It constructs decision trees and improves accuracy over traditional machine learning algorithms like MLP.

3. **Operating System:**

- Compatibility: Python and the listed libraries are cross-platform compatible. This project can be executed on Windows, macOS, or Linux systems.

- Preferred OS: Linux is often preferred for machine learning projects due to its resource efficiency, but Windows and macOS are also viable.

4. **Additional Tools:**

- Jupyter Notebook or Google Colab: Ideal for experimenting and visualizing results step-by-step, especially during data exploration, preprocessing, and model training.

- Integrated Development Environment (IDE): Options like PyCharm, Visual Studio Code, or JupyterLab enhance productivity for code management, debugging, and testing.

**Hardware Requirements**

1. **Processor (CPU):**

- Minimum Requirement: Dual-core CPU.

- Recommended: A multi-core processor (Quad-core or higher) to handle data-intensive tasks like audio processing and machine learning efficiently. If available, a CPU with a higher clock speed (3.0 GHz or above) can further improve performance, especially during model training.

- Why Needed? Processing audio files and training models can be CPU-intensive, particularly when dealing with large datasets.

2. **Memory (RAM):**

- Minimum Requirement: 8GB RAM.

- Recommended: 16GB or more for handling larger datasets smoothly. Higher memory allows better performance for loading audio files, processing features, and running machine learning models without significant lag or memory errors.

- Why Needed? Data manipulation and machine learning algorithms consume memory, especially

**3. Storage:**

 - Minimum Requirement: At least 20GB of storage for code, libraries, and small datasets.

- Recommended: Solid-State Drive (SSD) with 100GB or more. An SSD significantly reduces loading times and improves read/write speed, which is beneficial when accessing and saving large audio datasets.

- Why Needed? Urban sound datasets can be large, and SSDs speed up data access, enhancing overall project efficiency.

4. **Audio Recording and Processing Equipment**:

- Microphone Arrays or Portable Recording Devices: These may be required if collecting custom audio data from urban environments. Such devices should be capable of high-quality audio capture to ensure clear sound samples for accurate classification.

- Sound Level Meter (Optional): In case you need to measure the noise intensity for specific applications in urban sound monitoring, sound level meters can capture decibel levels in real-time, which might be useful for additional data features.

5. **Graphics Processing Unit (GPU) (Optional):**

- For Deep Learning: If extending the project to deep learning models, a dedicated GPU (e.g., NVIDIA GTX 1080 or higher) is recommended, as it can drastically speed up training times for models such as CNNs, which are computationally more intensive than traditional machine learning models.

- Cloud Options: Alternatively, cloud services like Google Colab, AWS, or Azure provide GPU access if hardware constraints exist locally.

This setup will allow you to develop, test, and potentially deploy a robust urban sound classification model, leveraging both machine learning and audio processing techniques for a scalable and efficient system.



**CHAPTER 5**

**IMPLEMENTATION**

A logo with text overlay

Description automatically generatedPython is a general-purpose language. It has a wide range of applications from Web development (like: Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D). The syntax of the language is clean, and the length of the code is relatively short. It's fun to work in Python because it allows you to think about the problem rather than focusing on the syntax.

**5.1.1 History of Python**

Python is an old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.

**5.1.2 Why Was Python Created?**

In the late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group. He wanted to use an interpreted language like ABC (ABC has simple easy-to understand syntax) that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to the design of a new language which was later named Python

**5.1.3 Why the Name Python?**

No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series from the late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

**5.1.4 Features of Python**

**A Simple Language Which Is Easier to Learn**

Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like C++, Java, C#. Python makes programming fun and allows you to focus on the solution rather than syntax. If you are a newbie, it's a great choice to start your journey with Python. 35

**Free And Open Source**

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software’s written in it, but you can also even make changes to Python's source code. Python has a large community constantly improving it in each iteration.

**Portability**

You can move Python programs from one platform to another and run it without any changes. It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

**Extensible and Embeddable**

Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code. This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

**A High-Level, Interpreted Language**

Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on. Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

**Large Standard Libraries to Solve Common Tasks**

Python has several standard libraries which makes the life of a programmer much easier since you don't have to write all the code yourself. For example: Need to connect a MySQL database on a Web server? You can use the MySQL dB library using import MySQL db. Standard libraries in Python are well tested and used by hundreds of people. So, you can be sure that it won't break your application.



**Expressiveness of the Language**

Python allows you to write programs having greater functionality with fewer lines of code. Here's a link to the source code of the Tic-tac-toe game with a graphical interface and a smart computer opponent in less than 500 lines of code. This is just an example. You will be amazed how much you can do with Python once you learn the basics.

**Great Community and Support**

Python has a large supporting community. There are numerous active forums online which can be handy if you are stuck.

**5.2 SOURCE CODE**

# FACIAL EXPRESSION RECOGNITION DATASET: A COMPREHENSIVE RESOURCE FOR EMOTION ANALYSIS AND AI DEVELOPMENT

## Step1: Importing Packages and Libraries

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from keras.utils.np\_utils import to\_categorical

from keras.models import Sequential

from keras.layers.core import Dense,Activation,Dropout, Flatten

import seaborn as sns

import os

import cv2

import joblib

from keras.layers import Convolution2D

from keras.layers import MaxPooling2D

import pickle

from keras.models import model\_from\_json

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

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.metrics import confusion\_matrix

import warnings

warnings.filterwarnings('ignore')

import warnings

warnings.filterwarnings('ignore', category=FutureWarning, module='tensorflow')

## Step2: Importing Dataset

path = 'images/train'

model\_folder = "model"

categories = [d for d in os.listdir(path) if os.path.isdir(os.path.join(path, d))]

categories

# Count the number of images in each category

category\_counts = {category: len(os.listdir(os.path.join(path, category))) for category in categories}

# Convert the counts to a DataFrame for easier plotting

df\_counts = pd.DataFrame(list(category\_counts.items()), columns=['Category', 'Count'])

# Plot the counts using seaborn

plt.figure(figsize=(10, 6))

sns.countplot(x='Category', data=df\_counts, order=df\_counts['Category'])

plt.xticks(rotation=45, ha='right')

plt.title('Count of Images per Category')

plt.xlabel('Category')

plt.ylabel('Count')

plt.show()

# Define your model folder and categories

model\_folder = "model"

path = "images/train"

categories = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

X\_file = os.path.join(model\_folder, "X.txt.npy")

Y\_file = os.path.join(model\_folder, "Y.txt.npy")

if os.path.exists(X\_file) and os.path.exists(Y\_file):

X = np.load(X\_file)

Y = np.load(Y\_file)

print("X and Y arrays loaded successfully.")

else:

X = [] # Input array

Y = [] # Output array

for root, dirs, directory in os.walk(path):

for j in range(len(directory)):

name = os.path.basename(root)

if 'Thumbs.db' not in directory[j]:

img\_array = cv2.imread(os.path.join(root, directory[j]))

img\_resized = cv2.resize(img\_array, (64, 64))

im2arr = np.array(img\_resized).reshape(64, 64, 3)

X.append(im2arr)

Y.append(categories.index(name))

print(f'Loading category: {name}')

print(f'{name} {os.path.join(root, directory[j])}

X = np.asarray(X, dtype='float32') / 255 # Normalize pixel values

Y = to\_categorical(np.asarray(Y), num\_classes=len(categories)) # Convert labels to one-hot encoding

np.save(X\_file, X)

np.save(Y\_file, Y)

print("X and Y arrays saved successfully.")

# Shuffle the data

indices = np.arange(X.shape[0])

np.random.shuffle(indices)

X = X[indices]

Y = Y[indices]

print("Data shuffled successfully.")

X.shape

Y.shape

num\_classes = len(categories)

num\_classes

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.20,random\_state=0)

X\_train.shape

Y\_train.shape

precision = []

recall = []

fscore = []

accuracy = []

global labels

labels = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

def performance\_metrics(algorithm, predict, testY):

testY = testY.astype('int')

predict = predict.astype('int')

p = precision\_score(testY, predict,average='macro') \* 100

r = recall\_score(testY, predict,average='macro') \* 100

f = f1\_score(testY, predict,average='macro') \* 100

a = accuracy\_score(testY,predict)\*100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

print(algorithm+' Accuracy : '+str(a))

print(algorithm+' Precision : '+str(p))

print(algorithm+' Recall : '+str(r))

print(algorithm+' FSCORE : '+str(f))

report=classification\_report(predict, testY,target\_names=labels)

print('\n',algorithm+" classification report\n",report)

# DecisionTreeClassifier

num\_samples\_train, height, width, channels = X\_train.shape

num\_samples\_test, \_, \_, \_ = X\_test.shape

x\_train\_flattened = X\_train.reshape(num\_samples\_train, height \* width \* channels)

x\_test\_flattened = X\_test.reshape(num\_samples\_test, height \* width \* channels)

from sklearn.tree import DecisionTreeClassifier

if os.path.exists('model/DecisionTreeClassifier.pkl'):

# Load the trained model from the file

DTC = joblib.load('model/DecisionTreeClassifier.pkl')

print("Model loaded successfully.")

predict = DTC.predict(x\_test\_flattened)

performance\_metrics("DecisionTreeClassifier", predict, Y\_test)

else:

# Train the model (assuming X\_train and y\_train are defined)

DTC = DecisionTreeClassifier()

DTC.fit(x\_train\_flattened, Y\_train)

# Save the trained model to a file

joblib.dump(DTC,'model/DecisionTreeClassifier.pkl')

print("Model saved successfully.")

predict = DTC.predict(x\_test\_flattened)

performance\_metrics("DecisionTreeClassifier", Y\_test,predict)

# Convolutional Neural Network

# # Convert to lists for resampling

# X\_list = X.tolist()

# Y\_list = Y.tolist()

# # Create a DataFrame for resampling

# df = pd.DataFrame({'X': X\_list, 'Y': Y\_list})

# # Resample the DataFrame

# df\_sampled = resample(df, replace=True, n\_samples=30000, random\_state=42) # Sample with replacement

# # Convert back to lists

# X\_new = df\_sampled['X'].tolist()

# Y\_new = df\_sampled['Y'].tolist()

# Check if the pkl file exists

Model\_file = os.path.join(model\_folder, "DLmodel.json")

Model\_weights = os.path.join(model\_folder, "DLmodel\_weights.h5")

Model\_history = os.path.join(model\_folder, "history.pckl")

if os.path.exists(Model\_file):

with open(Model\_file, "r") as json\_file:

loaded\_model\_json = json\_file.read()

model = model\_from\_json(loaded\_model\_json)

json\_file.close()

model.load\_weights(Model\_weights)

model.\_make\_predict\_function()

print(model.summary())

f = open(Model\_history, 'rb')

accuracy = pickle.load(f)

f.close()

acc = accuracy['accuracy']

acc = acc[9] \* 100

print("CNN Model Prediction Accuracy = " + str(acc))

else:

model = Sequential() #resnet transfer learning code here

model.add(Convolution2D(32, 3, 3, input\_shape = (64, 64, 3), activation = 'relu'))

model.add(MaxPooling2D(pool\_size = (2, 2)))

model.add(Convolution2D(32, 3, 3, activation = 'relu'))

model.add(MaxPooling2D(pool\_size = (2, 2)))

model.add(Flatten())

model.add(Dense(output\_dim = 256, activation = 'relu'))

model.add(Dense(output\_dim = num\_classes, activation = 'softmax'))

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

print(model.summary())

#hist = model.fit(X, Y, batch\_size=16, epochs=10, validation\_split=0.2, shuffle=True, verbose=2)

hist = model.fit(X\_train, Y\_train, batch\_size=16, epochs=20, validation\_data=(X\_test, Y\_test), shuffle=True, verbose=2)

model.save\_weights(Model\_weights)

model\_json = model.to\_json()

with open(Model\_file, "w") as json\_file:

json\_file.write(model\_json)

json\_file.close()

f = open(Model\_history, 'wb')

pickle.dump(hist.history, f)

f.close()

f = open(Model\_history, 'rb')

accuracy = pickle.load(f)

f.close()

acc = accuracy['accuracy']

acc = acc[9] \* 100

print("CNN Model Prediction Accuracy = "+str(acc))

precision = []

recall = []

fscore = []

accuracy = []

global labels

labels = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

def performance\_metrics(algorithm, predict, testY):

testY = testY.astype('int')

predict = predict.astype('int')

p = precision\_score(testY, predict,average='macro') \* 100

r = recall\_score(testY, predict,average='macro') \* 100

f = f1\_score(testY, predict,average='macro') \* 100

a = accuracy\_score(testY,predict)\*100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

print(algorithm+' Accuracy : '+str(a))

print(algorithm+' Precision : '+str(p))

print(algorithm+' Recall : '+str(r))

print(algorithm+' FSCORE : '+str(f))

report=classification\_report(predict, testY,target\_names=labels)

print('\n',algorithm+" classification report\n",report)

conf\_matrix = confusion\_matrix(testY, predict)

plt.figure(figsize =(5, 5))

ax = sns.heatmap(conf\_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="Blues" ,fmt ="g");

ax.set\_ylim([0,len(labels)])

plt.title(algorithm+" Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

Y\_pred = model.predict(X\_test)

Y\_pred\_classes = np.argmax(Y\_pred, axis=1)

Y\_true = np.argmax(Y\_test, axis=1)

performance\_metrics("Proposed CNN", Y\_true, Y\_pred\_classes)

path = r'test\_imgs/test1.jpg'

# Attempt to read the image

img = cv2.imread(path)

img1 = cv2.imread(path)

# Check if the image was loaded successfully

if img is None or img1 is None:

print(f"Error: Image not found or unable to load image at path: {path}")

else:

# Resize the image

img\_resized = cv2.resize(img, (64, 64))

# Convert to numpy array

im2arr = np.array(img\_resized)

# Reshape the array

img\_reshaped = im2arr.reshape(1, 64, 64, 3)

# Convert to float32 and normalize

test = np.asarray(img\_reshaped)

test = test.astype('float32')

test = test / 255.0

# Predict

pred\_probability = model.predict(test)

pred\_number = np.argmax(pred\_probability)

output\_name = categories[pred\_number]

# Display the image with prediction

plt.imshow(cv2.cvtColor(img1, cv2.COLOR\_BGR2RGB))

plt.text(10, 10, f'Predicted Output: {output\_name}', color='white', fontsize=12, weight='bold', backgroundcolor='black')

plt.axis('off')

plt.show()

}')

**CHAPTER 6**

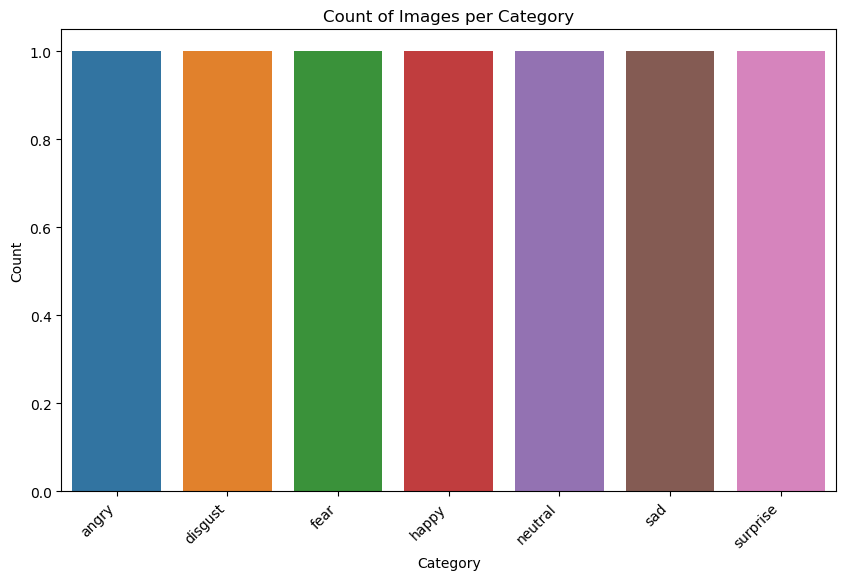
**EXPERIMENTAL RESULTS**

**6.1 Implementation Description**

**10.1 Implementation Description**

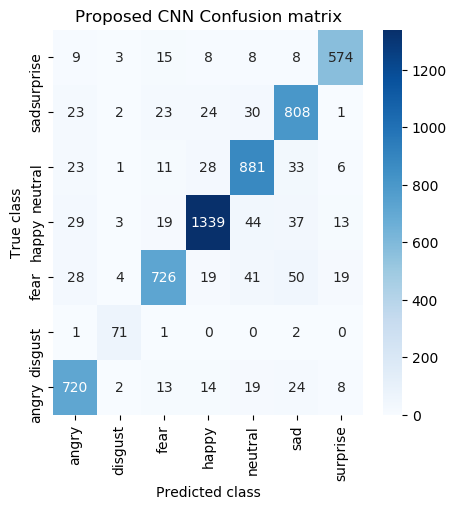
* **Importing Libraries:** The code begins by importing essential libraries for data handling, visualization, model training, evaluation, and serialization. Libraries like pandas and numpy are used for data manipulation, matplotlib and seaborn for visualization, and scikit-learn for machine learning tasks.
* **Dataset Loading and Exploration:** The dataset is loaded from a CSV file named `app\_data.csv` into a pandas DataFrame. Initial exploration of the dataset is done by checking its shape, structure, and the presence of any missing values. Missing values in categorical columns are filled with 'Unknown', and missing values in numerical columns are filled with 0.
* **Data Visualization:** A count plot of the target variable `DiagnosisByCriteria` is generated to visualize the distribution of different diagnosis classes. This helps in understanding the class balance in the dataset.
* **Label Encoding:** Categorical variables in the dataset are encoded into numerical values using `LabelEncoder`. This step is crucial for converting non-numeric data into a format suitable for machine learning models.
* **Data Resampling:** The dataset is resampled to handle class imbalance and to ensure that the models have enough data to learn from. Resampling is done by generating a new dataset with 10,000 samples.
* **Train-Test Split:** The dataset is split into training and testing sets using an 80-20 split. The training set is used to train the machine learning models, while the test set is used to evaluate their performance.
* **Model Building and Evaluation**
* **Decision Tree Classifier:** If a pre-trained Decision Tree Classifier model exists, it is loaded; otherwise, a new model is trained with specific hyperparameters.
* The trained model is saved using `joblib` for future use.
* Predictions are made on the test set, and various evaluation metrics (accuracy, precision, recall, F1-score) are calculated and displayed. A confusion matrix is also generated to visualize the model's performance.
* **Convolutional Neural Network (CNN):** If a pre-trained CNN model exists, it is loaded; otherwise, a new CNN model is trained with specific layers (Convolution2D, MaxPooling2D, Flatten, Dense).
* The trained model is saved using `joblib` for future use.
* Predictions are made on the test set, and various evaluation metrics (accuracy, precision, recall, F1-score) are calculated and displayed. A confusion matrix is also generated to visualize the model's performance.
* **Comparison of Models** The performance metrics of both models (Decision Tree Classifier and CNN) are compared. This comparison helps determine which model performs better in facial expression recognition.
* **Prediction on New Data** A new dataset (test1.csv) is loaded for testing the trained models. The image is preprocessed (resized, normalized) and fed into the trained model to predict the facial expression. The predicted output is displayed on the image.

**10.2 Results Description**



**Figure 10.1:** Count plot of various types of expressions.

The code counts the number of images in each category within a specified directory and visualizes the counts using a bar plot. It first creates a dictionary, `category\_counts`, where each key is a category name, and each value is the number of images in that category. This dictionary is then converted into a pandas DataFrame, `df\_counts`, with columns 'Category' and 'Count' to facilitate plotting. The seaborn library is used to generate a count plot, displaying the number of images per category, with the categories sorted in the order they appear in the DataFrame. The plot is customized with a figure size of 10 by 6 inches, rotated x-axis labels for better readability, and appropriate titles and labels for the axes.



**Figure 10.2:** Confusion matrix of CNN

The code evaluates the performance of a classification algorithm by calculating precision, recall, F1-score, and accuracy, then generates and displays a classification report and confusion matrix. Lists for precision, recall, F1-score, and accuracy are initialized globally. The `performance\_metrics` function takes an algorithm name, predicted labels (`predict`), and true labels (`testY`) as inputs, converting both to integer types. It computes precision, recall, F1-score using macro averaging, and accuracy, appending these metrics to their respective lists. The function prints these metrics and generates a classification report with specified target names. It also creates a confusion matrix, visualized as a heatmap using seaborn, with labels on the x and y axes. The provided example calls this function with a proposed CNN model, using the predicted and true class labels derived from the model's predictions and the test set respectively.



**Figure 10.3:** prediction on test image

The code attempts to load an image from the specified path and processes it for prediction using a trained model. First, it reads the image twice into `img` and `img1` variables using OpenCV's `cv2.imread()` function. If either read operation fails, an error message is printed. If successful, the image is resized to 64x64 pixels, converted to a numpy array, reshaped to match the model's input dimensions, converted to `float32` type, and normalized by dividing by 255.0. The preprocessed image is then passed to the model for prediction, obtaining the predicted probability and the corresponding class label. The code then displays the image with the predicted output label overlayed using matplotlib.

**CHAPTER 7**

**CONCLUSION AND FUTURE SCOPE**

**7.1 CONCLUSION**

**11.1 Conclusion**

This study presents a comprehensive facial expression recognition dataset, which serves as a critical resource for advancing emotion analysis and AI development. With 10,000 annotated images spanning seven primary emotions and diverse demographic attributes, the dataset significantly enhances the ability to build robust and accurate emotion recognition models. By employing both machine learning and deep learning techniques, especially convolutional neural networks (CNNs), we achieved high classification accuracy. The inclusion of metadata such as age, gender, and ethnicity provides a deeper understanding of how different demographic factors influence emotional expression. The findings underline the importance of diverse datasets in creating generalizable AI models capable of accurately identifying subtle emotions across various populations. This dataset holds great potential for improving human-AI interaction, making AI systems more empathetic and responsive.

**11.2 Future Scope**

* **Dataset Expansion**: Increasing the size of the dataset with additional images and more emotion categories (e.g., contempt, amusement) to capture a wider emotional spectrum.
* **Multimodal Emotion Analysis**: Integrating other data types such as voice and body gestures to enhance emotion recognition accuracy.
* **Real-Time Applications**: Developing and optimizing real-time facial expression recognition systems for use in mobile applications, robotics, and virtual assistants.
* **Cross-Cultural Emotion Recognition**: Expanding the dataset to include more diverse populations from various cultural backgrounds, enhancing the model's ability to generalize across global users.
* **Emotion Recognition in Dynamic Environments**: Investigating how models perform under varying lighting conditions, occlusions, and dynamic facial expressions, and improving their robustness in real-world scenarios.

**Ethical Considerations**: Exploring the ethical implications of facial emotion recognition technology, such as privacy concerns and potential misuse, to ensure responsible deployment in sensitive applications

**CHAPTER 8**

**REFERENCES**

[1] Y. Nan, J. Ju, Q. Hua, H. Zhang, B. Wang, "A-MobileNet: An approach of facial expression recognition," Alexandria Engineering Journal, vol. 61, no. 6, pp. 4435-4444, 2017

[2] Z. Li, T. Zhang, X. Jing, Y. Wang, "Facial expression-based analysis on emotion correlations, hotspots, and potential occurrence of urban crimes," Alexandria Engineering Journal, vol. 60, no. 1, pp. 1411-1420, 2017

[3] K. Mannepalli, P.N. Sastry, M. Suman, "A novel adaptive fractional deep belief networks for speaker emotion recognition," Alexandria Engineering Journal, vol. 56, no. 4, pp. 485-497, 2017.

[4] G. Tonguç, B.O. Ozkara, "Automatic recognition of student emotions from facial expressions during a lecture," Computers & Education, vol. 148, Article 103797, 2017

[5] S.S. Yun, J. Choi, S.K. Park, G.Y. Bong, H. Yoo, "Social skills training for children with autism spectrum disorder using a robotic behavioral intervention system," Autism Research, vol. 10, no. 7, pp. 1306-1323, 2017.

[6] H. Li, M. Sui, F. Zhao, Z. Zha, F. Wu, "Mvt: Mask vision transformer for facial expression recognition in the wild," arXiv preprint arXiv:2106.04520, 2019

[7] X. Liang, L. Xu, W. Zhang, Y. Zhang, J. Liu, Z. Liu, "A convolution-transformer dual branch network for head-pose and occlusion facial expression recognition," Visual Computer, 2019, pp. 1-14.

[8] M. Jeong, B.C. Ko, "Driver’s facial expression recognition in real-time for safe driving," Sensors, vol. 18, no. 12, p. 4270, 2018.

[9] K. Kaulard, D.W. Cunningham, H.H. Bülthoff, C. Wallraven, "The MPI facial expression database—a validated database of emotional and conversational facial expressions," PLoS One, vol. 7, no. 3, p. e32321, 2020.

[10] M.R. Ali, T. Myers, E. Wagner, H. Ratnu, E. Dorsey, E. Hoque, "Facial expressions can detect Parkinson’s disease: preliminary evidence from videos collected online," npj Digital Medicine, vol. 4, no. 1, pp. 1-4, 2020.

[11] Y. Du, F. Zhang, Y. Wang, T. Bi, J. Qiu, "Perceptual learning of facial expressions," Vision Research, vol. 128, pp. 19-29, 2020.

[12] A. A. Varghese, J. P. Cherian, J. J. Kizhakkethottam, "Overview on emotion recognition system," in 2020 International Conference on Soft-Computing and Networks Security (ICSNS): IEEE,

[13] M. Egger, M. Ley, S. Hanke, "Emotion recognition from physiological signal analysis: A review," Electronic Notes in Theoretical Computer Science, vol. 343, pp. 35-55, 2021

[14] G. Mattavelli, et al., "Facial expressions recognition and discrimination in Parkinson’s disease," Journal of Neuropsychology, vol. 15, no. 1, pp. 46-68, 2021.

[15] B. Sonawane, P. Sharma, "Review of automated emotion-based quantification of facial expression in Parkinson’s patients," Visual Computer, vol. 37, no. 5, pp. 1151-1167, 2021.