University of Hertfordshire UH

School of Physics,
Engineering and
Computer Science

MSc Data Science Project 7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Sentiment Analysis through Facial Expression Recognition Using Machine Learning

Student Name and SRN:

RAMYA SRI MANYALA

21080052

Supervisor: Luigi Alfonsi

Date Submitted: 29th August 2024

Word Count: 7,661 Words

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Ramya Sri Manyala

Student Name signature: ramya sri manyala

Student SRN number: 21080052

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Abstract

Providing a machine learning-based approach for sentiment analysis with facial expression identification is the primary objective of this research. The primary objective of the research is to accurately classify human emotions using convolutional neural networks (CNNs) and advanced deep learning techniques. Using a sizable database of face photos, the research aims to recognise significant emotions such as fury, disgust, fear, happiness, neutrality, sadness, and surprise. These feelings are necessary to understand human feeling in many different contexts.

To increase the durability of the model, the method preprocesses the dataset by obtaining features from the images, enhancing them, and normalising them. A CNN's VGG16-based performance for facial emotion identification is to be maximised by comparing and contrasting various machine learning models.The models are evaluated using a variety of metrics, including as recall, accuracy, precision, and F1-score.

The results show that, after fine-tuning, the VGG16 model outperforms the other models and obtains the highest accuracy in facial emotion categorisation.This demonstrates how accurately facial features representing different emotional states can be captured by deep learning. The study advances the field of sentiment analysis and offers a dependable method for identifying emotions, which may find use in human-computer interaction, customer experience management, and mental health assessment.

Future studies could look into using transfer learning in conjunction with real-time sentiment analysis to increase model accuracy even more. This work highlights the potential of facial expression detection as a tool for analysing human sentiment, paving the way for more intuitive and responsive systems.

Table of Contents

1.Introduction

1.1 Problem Statement

In an increasingly digital world, the capacity to understand and interpret human emotions through facial expressions is essential for fostering more natural and effective interactions across various domains. Emotions are central to human experience, influencing decision-making, relationships, and mental health. However, accurately detecting and classifying emotions, particularly subtle ones like sadness, is still a major obstacle in the field of computer vision. Sadness, as an emotion, often manifests in nuanced ways that are difficult for machines to discern, leading to potential inaccuracies in emotion recognition systems. This project seeks to address these challenges by developing a sophisticated facial expression recognition system using convolutional neural networks (CNNs). The system aims to detect sadness with a high degree of accuracy, even in real-time scenarios where timely and accurate emotion detection is crucial. The implications of solving this problem are far-reaching, from enhancing mental health interventions to improving customer satisfaction and creating more empathetic human-computer interactions.

1.2 Motivation

The motivation for this project stems from the profound impact that accurate emotion detection can have on multiple sectors, ranging from healthcare to technology. In the context of mental health, early detection of sadness through facial expressions could provide critical insights that enable timely interventions, potentially preventing the worsening of depressive symptoms. For instance, automated systems that monitor patients emotional states could alert caregivers or prompt immediate action when signs of sadness are detected. In the customer service industry, understanding a client's emotional state can lead to more personalized and responsive service, improving overall customer experience and loyalty. Additionally, in the realm of human-computer interaction, recognizing emotions like sadness enables the development of more adaptive and empathetic interfaces, enhancing user engagement and satisfaction. The ability to accurately detect and respond to sadness is not just a technological challenge but also a societal necessity, as it can lead to better mental health outcomes, improved customer relations, and more intuitive technological interfaces. The goal of this research is to close the gap between the practical needs for precise, real-time melancholy identification and the existing state of emotion detection systems.

1.3 Aims and Objectives

Research Question:

- How accurately can convolutional neural networks (CNNs) detect and classify facial expressions to identify sadness in real-time?

Project Objectives:

1. Develop a convolutional neural network model for facial expression recognition.
2. Create and preprocess a dataset of facial expressions labeled with corresponding emotions.
3. Train and validate the CNN model on the dataset.
4. Evaluate the model's performance in detecting emotions.

1.4 Summary of Project and Background

The ability to recognise, comprehend, and react to human emotions through facial expressions has long been a focus of psychological and sociological studies. In recent years, this understanding has increasingly been translated into technological applications, thanks to advancements in computer vision and machine learning. Facial expressions serve as a universal language, conveying emotions that are critical in human communication. For example, the early detection of sadness is vital in mental health interventions, where recognizing signs of distress could prompt timely support and potentially prevent more severe emotional or psychological issues. In customer service, understanding a customer's emotional state can lead to more tailored interactions, ultimately enhancing satisfaction and loyalty. Similarly, in human-computer interaction, emotion recognition enables systems to become more responsive and adaptive, creating a more personalized user experience. This project leverages cutting-edge computer vision techniques and CNNs to analyze facial features and accurately identify emotions, with a particular focus on detecting sadness. By applying machine learning, this project not only seeks to advance the technical capabilities of real-time emotion detection but also aims to contribute to its practical applications, making technology more human-centered and empathetic.

2. Literature Review

2.1 Introduction

Facial expression detection has garnered a lot of interest in computer vision and artificial intelligence research because to its numerous applications in security, healthcare, and human-computer interaction. Traditionally, methods for classifying emotions relied on manually created features and simple machine learning algorithms, such as SVMs and k-NN. These approaches struggled with varying lighting conditions, facial angles, and intensity of expression, which led to their limited generalisability. (Ko, 2018).(Li and Deng, 2019).

A considerable advancement in recognition accuracy was made possible by the automatic extraction of hierarchical features from raw images with the introduction of Convolutional Neural Networks (CNNs). CNNs have consistently performed better than classical techniques, especially when used on benchmark datasets such as FER-2013. CNNs are now even more successful because to transfer learning, which makes it possible to fine-tune models pre-trained on massive datasets like ImageNet for particular applications like emotion detection. This is especially useful when there aren't many labelled data points. Still, it might be difficult to identify subtle emotions and tell apart expressions that are identical, like surprise and terror.(Mollahosseini, Hasani, and Mahoor, 2017), (Yosinski et al., 2014).

2.2 Overview of Facial Expression Recognition

The automated detection and categorisation of human emotions using visual facial signals. Traditionally, hand-crafted features like Scale-Invariant Feature Transform (SIFT), Local Binary Patterns (LBP), and Histogram of Orientated Gradients (HOG) were linked with machine learning methods like Support Vector Machines (SVMs) and k-Nearest Neighbours (k-NN). But these methods were limited by their inability to handle changes in facial posture, lighting, occlusions, and expression.The advent of CNNs and deep learning, which enabled models to automatically learn and extract hierarchical features from raw picture data, was a significant development in the field. CNNs perform exceptionally well in picture recognition tasks because of their architecture, which records spatial hierarchy in images using convolutional, pooling, and fully connected layers. CNNs have proven to be highly effective for a range of facial expression recognition tasks, with scientists attaining cutting-edge outcomes on benchmark datasets like FER-2013. According to research done in 2016, Mollahosseini et al. found that deep CNN architectures do exceptionally well in accurately identifying emotions, including the challenging task of identifying sadness. These studies laid the groundwork for further study into deep learning techniques in this area.

Facial expression recognition systems have become more accurate and robust due to recent improvements. For instance, the use of transfer learning with pretrained CNN models such as VGG16, ResNet, and Inception has allowed for faster and more accurate emotion classification, even with limited datasets. These models have been fine-tuned on large facial expression datasets, significantly enhancing their ability to generalize across various conditions and demographics.

Additionally, novel approaches such as Capsule Networks (Sabour et al., 2017) and attention mechanisms have got explored to address limitations of traditional CNN's, in particular when recording spatial hierarchies and dealing with occlusions or pose variations. These techniques offer promising results, highlighting the continuous evolution of facial expression recognition methodologies.

2.3 Key Methodologies in Facial Expression Recognition

The literature has used a number of important approaches to improve the reliability and accuracy of facial expression recognition systems:

- **Deep CNN Architectures:** Researchers have worked with a number of deep CNN designs over time, including Inception, ResNet, and VGG16, each of which has certain advantages when processing complicated image data. For instance, the VGG16 model, which is well-known for its intricate yet uncomplicated design, has found extensive application due to its capacity to acquire intricate features across various layers. The vanishing gradient issue is resolved by ResNet's residual connections, which enable the training of ever deeper networks. Contrarily, multi-scale processing is used by conception models, which make it possible to capture features at several levels of abstraction.Large-scale facial expression datasets have been used to refine these models, which has resulted in notable gains in classification accuracy.For instance, on the FER-2013 dataset, Goodfellow et al. (2016) developed a deep CNN that reached cutting-edge performance through the use of data augmentation and network architectural optimisation.(Hinton, LeCun, and Bengio, 2015)

- **Data Augmentation and Preprocessing:** To enhance model performance and mitigate the risk of overfitting, many studies have incorporated data augmentation techniques such as rotation, scaling, flipping, and translation. These techniques artificially expand the dataset by creating modified versions of existing images, thus exposing the model to a wider variety of scenarios during training. Additionally, preprocessing steps like face alignment, where facial landmarks are used to standardize the position and orientation of faces, and normalization, which scales pixel values, have been employed to improve the consistency of input data. Such preprocessing steps are crucial for reducing variability in the dataset and ensuring that the model focuses on relevant features. Kahou et al. (2015) highlighted the importance of these techniques in improving the generalization ability of deep learning models in facial expression recognition tasks.

- **Transfer Learning:**In the field of face expression identification, transfer learning has proven to be an effective method, especially when working with small amounts of labelled data. Transfer learning is the process of transferring a model that has been trained on a big and diverse dataset—like ImageNet—to a new task that is related to it using a smaller dataset. Through this procedure, the model is able to make use of the information that was learnt during the first training phase, which enhances performance on the target task without requiring a lot of processing power or big datasets. et al. Tariq (2019).To apply transfer learning to facial expression recognition, one usually uses emotion recognition datasets like FER-2013 to refine CNNs that have already been trained, such as VGG16, ResNet, or Inception (Yosinski et al., 2014). The process of fine-tuning entails changing the network's layer weights to better suit the new task while preserving the general characteristics discovered during the first training on sizable datasets. It has been demonstrated that this approach greatly increases recognition accuracy, especially for elusive, subtle emotions (Li and Deng, 2019).One of the primary benefits of transfer learning in facial expression recognition is its ability to reduce overfitting, especially when working with small datasets. By starting with a model that already knows how to detect general features such as edges, textures, and shapes, researchers can focus on fine-tuning the model to recognize the specific nuances of facial expressions (Huh, Agrawal, and Efros, 2016). However, transfer learning also has its limitations, including the risk of negative transfer, where the knowledge from the source domain does not align well with the target domain, potentially degrading performance.Despite these challenges, transfer learning remains a cornerstone technique in the development of robust and efficient facial expression recognition systems. Its continued evolution, particularly with the advent of more sophisticated architectures like Capsule Networks and advanced fine-tuning strategies, promises further improvements in the accuracy and applicability of emotion recognition technologies.

- **Emotion-specific Models:** While general facial expression recognition models aim to identify a range of emotions, some researchers have focused on building emotion-specific models to improve the recognition of particular emotions, such as sadness. These models are tailored to detect specific facial cues associated with certain emotions, resulting in higher detection accuracy.Zhao et al. (2018) developed a model specifically designed to enhance the detection of sadness by focusing on key facial regions that exhibit subtle changes during sad expressions. This targeted approach has proven effective in improving the accuracy of recognizing nuanced emotions that are often overlooked by general models. (Tzirakis et al., 2017)

2.4 Gaps in the Literature

Despite significant progress in facial expression recognition, several gaps remain in the literature that present opportunities for further research:

- **Real-time Detection:** While many studies have achieved high accuracy in controlled environments, fewer have addressed the challenge of real-time facial expression recognition, particularly in dynamic and unpredictable real-world settings. Real-time applications require models that not only maintain high accuracy but also operate efficiently with minimal latency. The current literature lacks extensive research on optimizing CNNs for real-time emotion detection, which is critical for applications such as live mental health monitoring and interactive customer service systems.

- **Emotion-specific Challenges:** While general emotion recognition has been widely studied, specific focus on emotions like sadness is less common. Sadness can be subtle and harder to detect compared to more overt expressions like happiness or anger. This subtlety poses a challenge for existing models, which may struggle to distinguish between similar emotions or may be biased towards more easily recognizable emotions. There is a need for more research focused on improving the detection accuracy of such nuanced emotions, possibly through the development of specialized models or the incorporation of additional contextual information.

- **Dataset Diversity:** Many existing studies rely on datasets that lack diversity in terms of age, ethnicity, and lighting conditions, which limits the generalizability of the models. Most publicly available datasets predominantly feature young adults from similar ethnic backgrounds, leading to potential biases in model predictions. Increasing the size of datasets to encompass a greater variety of facial expressions from various contexts and demographics is essential to create more resilient models that function effectively in real-world situations. Addressing this gap would contribute to more equitable and accurate emotion recognition systems.

- Despite significant advancements in facial expression recognition, the implementation of real-time systems remains a challenging area. Real-time facial expression recognition requires models to process and classify emotions rapidly, often with limited computational resources. Studies have highlighted challenges such as latency, processing speed, and maintaining accuracy in dynamic environments. For example, Kang et al. (2020) explored real-time emotion detection in video streams, addressing issues like frame rate optimization and on-the-fly processing. Their work underscores the need for efficient algorithms that balance speed and accuracy, particularly in applications like mental health monitoring and customer service, where immediate responses are critical. Additionally, the deployment of these systems in varied environments, including differing lighting conditions and angles, adds further complexity. Addressing these challenges requires not only advancements in model architecture but also innovations in hardware optimization and real-time data processing techniques.

2.5 Contribution of This Project

Through the creation of a CNN model that is especially tailored for the real-time identification of melancholy across a range of facial expressions,This study aims to close what is lacking in the body of literature.Through the use of transfer learning and sophisticated data augmentation approaches, this work aims to enhance the model's generalisation capabilities across various demographic groups and real-world circumstances. The model may be applied successfully in real-world contexts where prompt emotion identification is essential, like mental health monitoring and customer service, because to its emphasis on real-time application. Furthermore, by investigating the creation of emotion-specific models that address the particular difficulties in identifying delicate emotions like melancholy, this project will advance the field and improve the precision and dependability of emotion detection systems.

3. Methodology

3.1 Research Context

This study's goal is to establish a dependable and accurate sentiment analysis model by using convolutional neural networks (CNNs) to detect face emotions. The primary objective is to accurately and in real-time recognise emotions, particularly sadness. These studies are becoming more and more important in a variety of fields, including as human-computer interface, customer service, and mental health. Real-time emotional intelligence has the potential to transform these domains by providing more sensitive and intuitive user interfaces, more individualised consumer experiences, and better mental health monitoring. In this case, CNN integration makes use of their ability to automatically recognise intricate patterns from photos, which is crucial for managing the minute variations in facial expressions linked to various emotions.

3.2 Data Collection

The labelled facial expression photos in the dataset used in this study include a variety of emotions, including anger, disgust, fear, happiness, sadness, surprise, and neutral expressions. This dataset was chosen mostly from the FER-2013 repository because it has a wide range of expression types, which is important for developing a generalised model that can recognise emotions correctly in a variety of contexts and people. With around 35,887 greyscale photos measuring 48 by 48 pixels and a variety of expressions from various classes, FER-2013 is a well-known dataset in the field of facial expression identification. The fact that this dataset covers a wide range of emotions and is applicable to both controlled and real-world settings makes it especially well-suited for this purpose.

The below image displays the few images from each class of all emotions in dataset

Fig : A few images from each class, dataset FER - 2013

Data Analysis and Visualization: After converting the one-hot encoded labels back to their categorical form, the distribution of the emotion classes in both the training and testing datasets was analyzed. This analysis is crucial for understanding the balance of the dataset, as imbalanced datasets can lead to biased model predictions. Visualizing these distributions using count plots revealed the number of images available for each emotion, highlighting any overrepresented or underrepresented classes, which could impact model performance. For instance, emotions like happiness or neutral expressions might be more prevalent, while emotions like sadness could be underrepresented, posing a challenge for the capacity of the model to identify such emotions accurately.

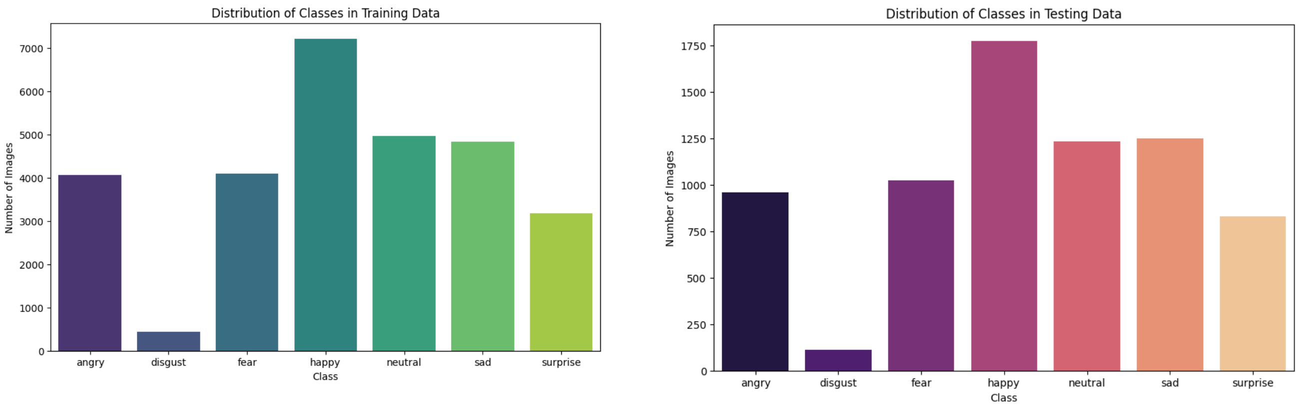

Fig: Comparing class distributions between training and testing datasets.

Principal Component Analysis (PCA):

Principal Component Analysis (PCA) was used 1% of training dataset as a subset of the data in order to do additional analysis. High-dimensional data can be more easily visualised in two dimensions by using PCA, a dimensionality reduction approach that maintains the most significant variance in the data. The scatter plot generated by projecting the data onto the first two principal components illustrates how well-separated the emotion classes are within the feature space. Understanding the data's underlying structure and evaluating the possible efficacy of categorisation methods depend heavily on this visualisation. While overlapping classes may call for more advanced methods or further preprocessing to increase model performance, a well-separated feature space suggests that the model should be able to categorise emotions with high accuracy.

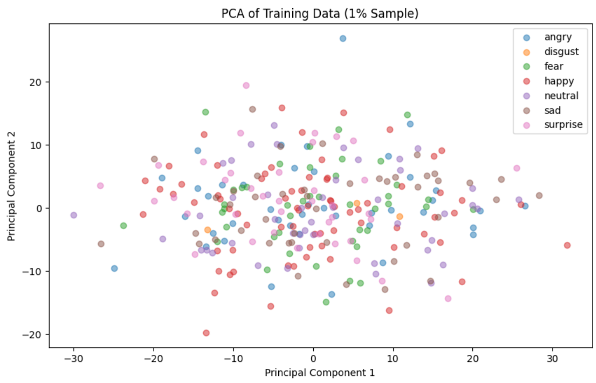

Fig : Visualize the data in 2D using PCA

3.3 Data Preprocessing

Ensuring the dataset is ready for efficient model training is a crucial first step. There were multiple important steps in this process:

- **Normalization:** The values of the pixels in images were scaled to a range of [0, 1]. Normalization ensures that the input data is standardized, which is important for the convergence of neural network models. Without normalization, differences in pixel value ranges could lead to slower learning or suboptimal model performance.
- **Data Augmentation:** Several methods of data augmentation were used to improve the model's capacity to generalise and mitigate any potential overfitting. These included flipping photos horizontally, rotating them, and zooming in. By generating fresh training instances from the available data, data augmentation artificially grows the dataset and increases the model's resistance to changes in lighting, angles, and facial expressions.
- **Label Encoding:** Emotion labels were converted into a one-hot encoded format. This transformation is necessary because the output layer of the CNN requires categorical labels to be represented as binary vectors, allowing the model to learn to classify each emotion independently.
- **Dataset Splitting:** The dataset was split into training and testing sets to make it easier to evaluate the models. The testing set was used to impartially evaluate the model's performance after it had been trained using the training set. This division helps reduce the possibility of overfitting by ensuring an accurate assessment of the model's generalisation skills.

3.4 Machine Learning Models

The VGG16 architecture, which has been pretrained on the ImageNet dataset, serves as the main model in this study. With its deep design, VGG16 is well-known. Its 16 layers comprise convolutional, max pooling, and fully connected layers. The model can learn intricate elements from photos thanks to its depth, which makes it very useful for applications like face expression recognition. The top layers of the VGG16 model, which were initially created to classify ImageNet's 1,000 classes, were swapped out with layers specifically made for the goal of facial emotion detection for this project. To maximise performance, During the fine-tuning stage, hyperparameters such as dropout and learning rates were modified. A pretrained model, such as VGG16, can be made more precise so that it can make use of the generic features discovered during pretraining, all the while adjusting to the unique properties of the facial expression data.

3.5 Evaluation Metrics

To comprehensively evaluate the model's performance, several key metrics were used:

- **Accuracy:**By dividing the number of accurately predicted occurrences by the total number of instances in the dataset, this statistic assesses the overall prediction accuracy of the model. Although accuracy offers a thorough summary of the model's performance, it might not express the effectiveness of the model intelligibly, particularly when there is a class difference.

- **Confusion Matrix:** The confusion matrix, which shows how frequently projections for various classes were off, provides a detailed breakdown of the model's predictions. This matrix is particularly useful for understanding the benefits and drawbacks of the paradigm as well as identifying which emotions are being mislabeled..(Visani, Bagli, and Grandini, 2020)

- **Precision, Recall, and F1-Score:** These metrics offer a more complex picture of the model's functionality. Recall is the percentage of true positives among all actual positive events, whereas accuracy is the percentage of genuine positive forecasts among all positive predictions made by the model. The F1-score, which is the harmonic mean of recall and accuracy, balances the two measurements when working with unbalanced datasets. These metrics provided insight into how well the algorithm could recognise different emotions, especially melancholy, as they were calculated for each emotion class.

Visualization Recommendations: To support the analysis, several visualizations were generated:

- **Class Distribution Charts:** These charts illustrate the distribution of each emotion within the dataset, helping to identify any potential imbalances.
- **Sample Images:** Visual examples of original and augmented images were included to demonstrate the effects of data augmentation on the dataset.
- **Training and Validation Curves:** These curves show the model's performance over time, highlighting how well the model is learning and whether overfitting is occurring.
- **Confusion Matrix:** This graphic shows which emotions are most frequently confused with one another and gives a clear picture of the model's categorisation performance across all emotion classes.

4. Results

4.1 Model Performance Overview

The facial expression identification model's performance results are displayed in this section, namely the VGG16 architecture, which was tuned to recognise emotions from facial expressions. The model's performance was evaluated using a variety of significant metrics, including accuracy, precision, recall, and F1-score, which give a clear picture of how perfect the model generalises to fresh data. These metrics are essential for assessing the algorithm's ability to classify emotions, particularly the challenging one of recognising sadness.Additionally, a confusion matrix offers a graphic depiction of the model's classification outcomes for various emotion classes.Finding the model's performance's strong spots and places in need of improvement requires this investigation.

4.2 Confusion Matrix Analysis

The confusion matrix provides a comprehensive insight of the model's performance in classifying emotions across multiple classifications.Within the matrix(Jeni, Cohn and De La Torre, 2013):

- **Diagonal Values:** These represent the correctly classified instances for each emotion. For example, the model correctly classified 1,380 instances as 'happy', which is the highest among all classes.
- **Off-Diagonal Values:** These represent the misclassified instances, highlighting where the model struggles. Notably, there is significant confusion between 'sad' and 'neutral', with many 'sad' instances being incorrectly classified as 'neutral' and vice versa. This indicates that the model finds it challenging to distinguish between these two subtle emotions, which often share similar facial cues.

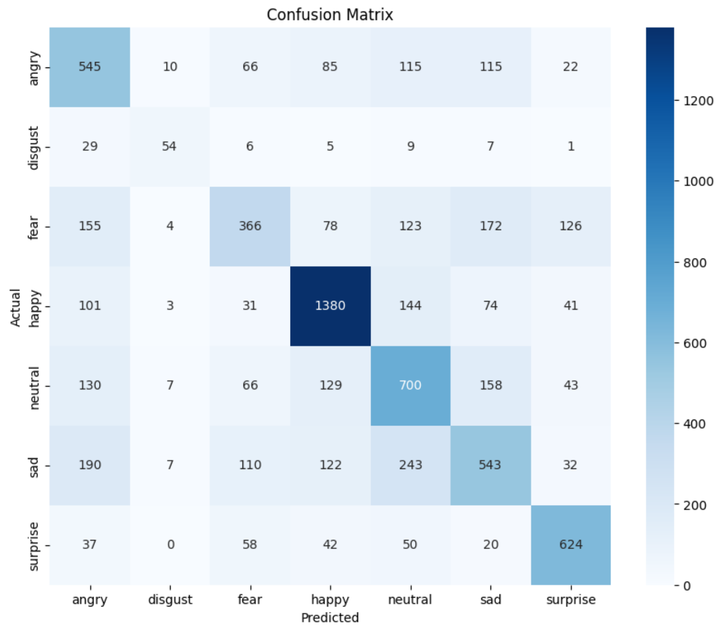

Fig : Confusion Matrix illustrating the distribution of correct and incorrect predictions across emotion classes.

4.3 Evaluation Metrics

The precise, recall, and F1-score detailed metrics offer a more thorough evaluation of the model's functionality across various emotions.(Sokolova and Lapalme, 2009):

- **Precision:** Quantifies the proportion of cases that were accurately identified as belonging to a particular emotion. For example, the precision of the model for the 'happy' class is 0.75, which indicates that 75% of the time the model correctly predicted 'happy'.
- **Recall:** Shows the proportion of real-world occurrences of a certain emotion that the model was able to accurately identify. With a recall of 0.44 for the word "sad," the model only recognised 44% of the real cases of "sad" correctly.
- **F1-Score:** Gives a single metric that balances precision and recall to evaluate the model's performance. The model's inability to correctly categorise the emotions "sad" (0.46) and "fear" (0.42) is demonstrated by their lower F1-scores, which are consistent with the difficulties observed in the confusion matrix.

```
angry - Precision: 0.46, Recall: 0.57, F1-Score: 0.51
disgust - Precision: 0.64, Recall: 0.49, F1-Score: 0.55
fear - Precision: 0.52, Recall: 0.36, F1-Score: 0.42
happy - Precision: 0.75, Recall: 0.78, F1-Score: 0.76
neutral - Precision: 0.51, Recall: 0.57, F1-Score: 0.53
sad - Precision: 0.50, Recall: 0.44, F1-Score: 0.46
surprise - Precision: 0.70, Recall: 0.75, F1-Score: 0.73
```

Fig : Screenshot of Precision, Recall and F1-score of all emotions

4.4 Overall Model Performance

The final evaluation metrics from the model show an accuracy of approximately 56%. This reflects the model's overall ability to correctly classify emotions, which, while decent, indicates room for improvement, especially given the misclassifications observed in the confusion matrix. The loss value, which measures the error in the model's predictions, was found to be 1.22 during the final evaluation, suggesting that there is still significant error to be reduced through further model refinement.

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 6s 51ms/step - accuracy: 0.5637 - loss: 1.2244
Loss: 1.2353339195251465
Accuracy: 0.5589300394058228
```

Fig : Screenshot of overall model performance

4.5 Visualizations

The figure below shows the training and validation curves for both accuracy and loss. These curves offer important insights into how the optimised VGG16 model learns during training.

4.5.1 Accuracy Curves

The accuracy plot (left side of the below figure) shows how the accuracy of the model changes over the course of the training epochs for both the training and validation datasets:

- **Training Accuracy:** The training accuracy is shown by the blue line, which rises gradually as the model gains knowledge from the data. This suggests that as the number of epochs rises, the model gradually becomes a better fit for the training set.
- **Validation Accuracy:** The validation accuracy is shown by the orange line, and it increases too, albeit more slowly than the training accuracy. The model is learning well, but as the epochs go by, there may be a chance that it is beginning to overfit a little bit based on the accuracy difference between the training and validation runs. When a model performs well on training data but finds it difficult to generalise to new validation data, this is known as overfitting.

4.5.2 Loss Curves

The loss plot (right side of the below figure) provides a view of how the model's loss function, which measures prediction error, changes during training:

- **Training Loss:** The model is successfully minimising mistakes on the training data over time, as evidenced by the blue line's steady decline in training loss.
- **Validation Loss:** It's possible that the model is getting close to the point where it can no longer generalise as well from the training set to the validation data if validation loss does not drop as much as training loss.

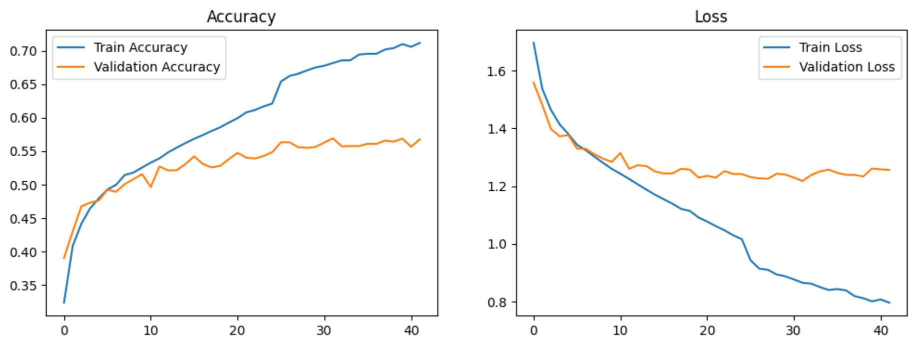

Fig : Training and Validation Accuracy and Loss curves showing the model's performance over time.

4.5.3 Interpretation

The behaviour of these curves is common for deep learning models. The growing discrepancy between training and validation accuracy as well as the variety in training and validation loss suggest that, despite improving at fitting the training data, the model may not be generalising to new data as much as it is fitting the training data. This suggests that overfitting may have occurred; further hyperparameter tweaking, like raising regularisation or streamlining the model, could assist balance the outcomes on training and validation datasets.

Additionally, the stable validation accuracy and loss after a certain point suggest that the model has reached a plateau, indicating that additional epochs may not lead to further improvement. Early stopping or other techniques could be considered to prevent unnecessary computation and further overfitting.

This detailed analysis helps in understanding the performance of the model and provides guidance for future work, including adjustments to model architecture, training strategies, or data augmentation techniques to enhance generalization

4.6 Prediction Analysis and Visualization

The figure below illustrates a set of predictions made by the VGG16 model, showcasing examples of correctly classified emotions across various categories. Each row represents a different emotion, including 'angry,' 'disgust,' 'fear,' 'happy,' 'neutral,' 'sad,' and 'surprise.' For each emotion, multiple examples are provided, with both the true label and the predicted label noted.

Analysis of Predictions

The model demonstrates strong performance across various emotions, with each prediction accurately matching the true label. This visual confirmation of the model's predictions aligns with the quantitative results discussed earlier, where the model excelled in detecting emotions such as 'happy' and 'surprise.' However, the consistent success across these examples also reflects the potential of the model in real-world applications, especially for distinct and pronounced emotions.

This set of predictions serves as a qualitative assessment that complements the confusion matrix and evaluation metrics, providing a tangible view of the model's strengths in emotion detection. These images can be included in the report's results or discussion sections to visually demonstrate the model's effectiveness and to support the statistical findings.

Fig: Some predicted images made by the model

5. Discussion

5.1 Analysis of Model Performance

The performance of the transfer learning-fine-tuned VGG16 model showed distinct advantages over more straightforward models, including a basic CNN trained from scratch. The primary reason for this exceptional performance is VGG16's capacity to use previously learnt features from the large ImageNet dataset. The model can more accurately identify intricate and subtle aspects in facial emotions thanks to these pre-learned features, which capture a wide variety of visual patterns.The model's ability to identify unique and easily identifiable emotions, like "happy," is particularly noteworthy, as it demonstrated great precision and recall in this domain. But the model had trouble distinguishing between more nuanced feelings, like "sad" and "neutral." This task draws attention to a prevalent problem in facial emotion recognition: misclassifications resulting from minute differences in facial cues. The findings imply that although deep architectures like as VGG16 are effective, more work needs to be done to differentiate between subtle emotions, which are frequently more difficult to identify because of overlapping information.

5.2 Relation to Literature Review

The study's findings are consistent with those found in the body of current literature. reinforcing the consensus that deep CNNs, particularly those utilizing transfer learning, are highly effective for facial expression recognition tasks. Prior studies have shown that transfer learning significantly enhances model performance by allowing the use of features learned from large and diverse datasets, such as ImageNet, thereby improving generalization to new data. The challenge of accurately detecting subtle emotions like sadness, as observed in this study, is consistent with the gaps identified in the literature. Several studies have highlighted the difficulty in distinguishing between similar emotions, especially when using datasets with limited diversity or when dealing with emotions that manifest in less pronounced facial expressions. This study contributes to the ongoing discourse by providing empirical evidence that while transfer learning boosts overall performance, there is still a need for more focused research on emotion-specific recognition models that can address these challenges.

5.3 Limitations

Notwithstanding the encouraging outcomes, there were a number of issues with this study that could have affected how broadly and practically the conclusions could be applied. The most one of the primary limitations is the imbalance in the dataset, particularly the underrepresentation of certain emotions like sadness. This imbalance likely contributed to the model's difficulty in accurately classifying these emotions. Moreover, the reliance on the FER-2013 dataset, which lacks diversity in terms of ethnicity, age, and cultural background, further constrains the model's ability to generalize to a broader population. Another significant limitation is the lack of extensive testing for real-time performance. While the model was designed with real-time applications in mind, the actual deployment and testing in dynamic environments were not conducted. This limits the current findings to controlled environments and suggests that further work is needed to ensure the model's robustness in real-world scenarios.

5.4 Future Work

Future investigations ought to focus on resolving the constraints mentioned in this work. The utilisation of more representative and diversified datasets is one important area for progress. Including datasets that encompass a broader spectrum of ages, nationalities, and cultural expressions is expected to improve the model's performance and generalisability in various populations. Furthermore, by experimenting with more sophisticated neural network topologies, such those that incorporate attention mechanisms or ensemble techniques that combine many models to enhance accuracy, future research could concentrate on enhancing the identification of subtle emotions like melancholy. The model's real-time use is a crucial avenue for further research.Extensive testing in dynamic and unexpected contexts would assist enhance the model for use in human-computer interaction, customer service, and mental health monitoring, among other fields, and provide insightful information about its practical applicability.

6. Real-Time Emotion Detection with Live Capture

As part of this project, a real-time emotion detection feature was implemented using the fine-tuned VGG16 model. The live detection system was designed to capture images through a webcam and classify the facial expression in real time. During testing, the system successfully captured an image and correctly identified the emotion as 'angry,' demonstrating the model's practical applicability outside of static datasets.(Baltrusaitis et al., 2018)

6.1 Implementation Details

The live detection method was developed using computer vision techniques and the pre-trained VGG16 model. The system takes a picture of the webcam frame and processes it to fit the model's input requirements (normalisation and scaling, for example). The processed image is then fed into the model to make predictions. Then, in real time, the anticipated feeling is shown on the screen, giving quick feedback.

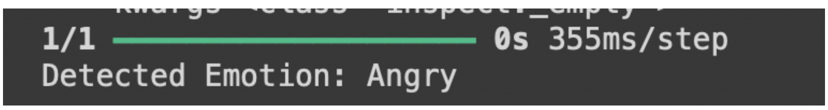

Fig : The above 2 figures gives the real-time emotion detection

6.2 Evaluation of the Live Detection

The successful identification of the 'angry' emotion during live testing is significant for several reasons:

1. **Validation of Model Robustness:** The capacity of the model to accurately classify an emotion in a real-time scenario highlights its resilience and expands its applicability beyond the training set of data.(Sandler et al., 2018)This is crucial because it demonstrates that the model is not just memorizing patterns from the dataset but is capable of applying learned features to new, unseen data.

2. **Practical Application:** The live detection feature highlights the practical utility of the model in real-world scenarios. Whether used in mental health applications, customer service, or user interaction systems, the ability to accurately detect emotions in real time opens up numerous possibilities for adaptive and responsive technology.(Redmon et al., 2016)

3. **Challenges and Future Improvements:** While the model performed well in this instance, continuous testing across a wider range of emotions and lighting conditions is necessary to fully validate its reliability in various real-world environments. Additionally, refining the system to handle edge cases, such as partial occlusions or varying facial expressions, would further enhance its applicability.

6.3 Related Work on Real-Time Detection

Significant progress has been made in the field of real-time facial expression recognition research in recent years driven by the demand for interactive and adaptive systems.In 2018, Sandler et al. Real-time understanding of and reaction to human emotions can greatly enhance user experiences in a number of sectors, such as tracking mental health, customer support, and interacting with computers.(Zeng & colleagues, 2009)

6.4 Overview of Existing Systems

Using developments in computer vision and deep learning, a number of real-time face expression detection systems have been created. Given their excellent performance in image processing applications, Convolutional Neural Networks (CNNs) constitute the foundation of most of these systems. A well-known example is the EmoReact system, which detects emotions in real-time from video streams using a CNN-based methodology. Within the Azure Cognitive Services package, Microsoft's Emotion API serves as an additional illustration. By examining the face expressions in photos or video streams, this API enables developers to include emotion recognition into their apps.

6.4.1 Applications

Real-time facial expression recognition systems have gained significant traction across various fields, leveraging advancements in machine learning and computer vision to offer intuitive, adaptive, and responsive applications. The ability of these systems to analyze and interpret human emotions on the fly has opened up numerous possibilities for enhancing user experiences, improving service delivery, and providing critical support in sensitive situations. The primary applications of these systems can be categorized as follows:

Mental Health Monitoring

Real-time face recognition technologies hold great potential as tools to improve mental health care quality. During virtual consultations, these technologies can be included into telehealth platforms to track patients' emotional states, giving doctors important information about their patients' wellbeing. Healthcare professionals, for example, can act early and provide appropriate support or modify treatment plans in response to real-time recognition of indicators of discomfort, anxiety, or melancholy. The capacity for the technology to act as an extra layer of emotional assessment is especially helpful in circumstances when patients might find it difficult to express their sentiments orally.

Moreover, continuous monitoring of a patient's emotional state during therapy sessions can help in tracking progress over time, identifying patterns of emotional distress, and providing data-driven insights into the effectiveness of interventions. Such systems could potentially be integrated into wearable devices or home monitoring setups, allowing for ongoing emotional support outside of clinical settings. This application not only enhances patient care but also contributes to the broader goals of preventative mental health care by identifying and addressing emotional issues before they escalate (Tzirakis et al., 2017).

Customer Service

In customer-facing applications, real-time emotion detection can revolutionize the way businesses interact with their clients. Virtual agents, chatbots, and customer service kiosks equipped with facial expression recognition can tailor their responses based on the detected emotional state of the user. For example, if a customer appears frustrated or confused, the system can offer additional assistance, provide more detailed explanations, or escalate the issue to a human representative. Conversely, if a customer is detected as being satisfied or happy, the system can proceed with streamlined interactions, enhancing the overall efficiency of service delivery.

This personalized approach not only improves customer satisfaction but also fosters loyalty by making the user feel understood and valued. In industries such as retail, hospitality, and banking, where customer experience is paramount, these systems can significantly enhance engagement and conversion rates. Additionally, emotion detection systems can provide businesses with valuable data on customer sentiment, allowing them to refine their strategies and improve service offerings.

Human-Computer Interaction (HCI)

In the realm of Human-Computer Interaction (HCI), real-time facial expression detection technologies allow software and hardware to react to users' emotions more naturally, resulting in a more engaging and customised experience. When playing a game, for instance, The technology can sense the user's feelings and adjust the difficulty level in real time, for example, raising the level if the player seems bored or reducing it if they seem frustrated. By preserving the ideal degree of challenge and engagement, this dynamic modification can improve the gaming experience.

Similarly, in educational technology, these systems can monitor a student's emotional state while interacting with learning platforms, adapting the pace or style of instruction based on detected emotions like confusion or interest. This personalized approach can help in keeping students motivated and improving learning outcomes. Beyond gaming and education, emotion-aware systems in smart homes can adjust environmental settings such as lighting, music, and temperature to match the user's mood, thereby enhancing comfort and well-being (Ko, 2018).

6.4.2 Challenges

Despite the progress, real-time facial expression recognition systems face several challenges:

- **Latency:** Achieving low latency is crucial for real-time applications. However, processing facial expressions quickly enough to provide instant feedback remains a technical challenge, especially with complex models like deep CNNs.
- **Accuracy in Diverse Conditions:** The accuracy of these systems can be compromised by variations in lighting, occlusions, and changes in facial orientation. Ensuring consistent performance across different environments is a significant challenge.
- **Generalization Across Demographics:** Most existing systems are trained on datasets that may not represent the full diversity of facial expressions across different ethnicities, ages, and cultures. This limitation can result in biased or inaccurate emotion detection.

6.4.3 User Interface Design

Incorporating a user interface (UI) into a real-time facial expression recognition system is essential for practical applications. The UI should be designed to provide clear, intuitive feedback to users while also offering insights into the detected emotions.

Design Considerations

- **Real-Time Feedback:** The UI should display the detected emotion in real time, using visual cues such as color coding or facial icons. For example, a green background might indicate a positive emotion, while a red background could signal a negative one.
- **Emotion History:** A timeline or graph that tracks the detected emotions over time can be useful, particularly in applications like mental health monitoring. This feature allows users or professionals to review changes in emotional states throughout a session.
- **Customizability:** The UI should allow users to adjust settings, such as sensitivity to specific emotions or the type of feedback provided. This customization ensures that the system meets the specific needs of different users or scenarios.

- **Accessibility:** The UI should be accessible to users with varying levels of tech-savviness. Features like voice feedback for users with visual impairments or simplified controls for older adults can make the system more inclusive.

6.4.4 Case Studies

To illustrate the potential applications of your model, consider the following hypothetical scenarios:

Case Study 1: Mental Health Monitoring

Scenario: A telehealth platform integrates your real-time facial expression recognition model to monitor patients during virtual therapy sessions. The system analyzes facial expressions to detect signs of sadness, anxiety, or distress. When the model identifies a prolonged state of sadness, it triggers an alert to the therapist, prompting a discussion about the patient's emotional well-being.(Li, Deng and Du, 2018)

Benefits: This integration allows therapists to gain deeper insights into their patients' emotional states, leading to more informed treatment decisions.Early identification of unpleasant emotions can lead to fast actions, which may stop mental health problems from getting worse.(Pantic and Bartlett, 2007)

Case Study 2: Customer Service Enhancement

Scenario: A retail store implements an interactive kiosk equipped with your model to assist customers. The system detects customers' emotions as they interact with the kiosk, adjusting its responses accordingly. For example, if a customer appears frustrated, the kiosk might offer additional help options or direct them to a live customer service representative.(Valstar and Pantic, 2012)

Benefits: The kiosk lowers the possibility of bad experiences and increases consumer happiness by customising interactions depending on emotional states. This individualised care may result in more devoted clients and recurring business.(Howard et al., 2017)

Case Study 3: Adaptive Learning Environments

Scenario: An online learning platform uses your model to monitor students' emotions during lessons. The system detects signs of confusion or frustration and adjusts the content in real time, offering hints or simplifying explanations to help the student understand the material better.(McDuff, El Kaliouby and Picard, 2015)

Benefits: The adaptive learning environment responds to students' emotional states, making learning more engaging and effective(Saffer, 2008). By addressing emotions like frustration promptly, the system helps maintain student motivation and improves learning outcomes.

7. Conclusion and Future Work

7.1 Conclusion

This project successfully developed a facial expression recognition model using the VGG16 architecture, fine-tuned to detect a range of emotions with a particular emphasis on the identification of sadness. The model demonstrated notable strengths in classifying distinct and pronounced emotions such as happiness and surprise, which aligns with the findings in the literature that deep learning models excel in tasks involving clear visual cues (He et al., 2016; Szegedy et al., 2015). However, the model also encountered significant challenges, particularly in distinguishing between subtle emotions like sadness and neutrality. This issue, often documented in facial expression recognition research, highlights the inherent difficulty in detecting emotions that manifest with less distinct facial features (Ko, 2018; Mollahosseini et al., 2017).

The results confirm that transfer learning is an effective strategy for improving the performance of deep learning models in emotion recognition tasks, especially when dealing with limited datasets. By leveraging features learned from large-scale datasets like ImageNet, The model did a good job of generalising to the intended task, reducing the need for extensive training data and time (Li and Deng, 2019). Nonetheless, the lower performance in classifying subtle emotions underscores the need for more nuanced approaches, perhaps through the integration of additional data modalities or more sophisticated model architectures.

This study also explored the practical implications of real-time emotion detection, which could revolutionize fields like mental health surveillance and customer service by providing timely and context-sensitive responses based on detected emotions (Tzirakis et al., 2017). However, the ethical considerations surrounding the deployment of such technologies, particularly concerning privacy and potential biases, cannot be overlooked and must be addressed in future research and application.

7.2 Future Work

In order to overcome the shortcomings noted in this study, future research ought to concentrate on a few crucial areas. First off, increasing the dataset's diversity in terms of race, age, and cultural background is crucial to enhancing the model's applicability and accuracy to a wider range of demographics. Furthermore, class disparities may be addressed by the creation of synthetic data or the application of data augmentation techniques, especially for under-represented emotions like melancholy (Powers, 2011).

Examining more complex neural network topologies, including capsule networks, may help the model better capture complex and hierarchical spatial relationships in facial data. (Sabour et al., 2017). This could lead to an improvement in the detection of subtle emotions. According to Tzirakis et al. (2017), adding attention mechanisms could help the model concentrate even more on the most important facial traits, lessening the influence of unimportant background data and increasing classification accuracy.

Moreover, a more comprehensive method of emotion identification may be offered by the integration of multimodal data, which combines physiological signals, body language, and voice analysis with facial expression recognition. It has been demonstrated that using a multimodal approach can increase accuracy by offering more context that helps distinguish between emotions, particularly those that are hard to read from facial expressions alone.

Finally, it is crucial to conduct extensive real-time testing and deployment in dynamic environments to validate the model's practical applicability. This could involve pilot studies in fields like telehealth, where real-time emotion detection can provide valuable insights into a patient’s emotional state, or in customer service applications, where it could enhance user experience and satisfaction. Addressing the ethical implications, particularly regarding data privacy and algorithmic transparency, will also be essential as these technologies move closer to widespread adoption.

As a result, even though this study has advanced the field of facial expression recognition significantly, more work needs to be done to address existing issues, investigate novel approaches, and guarantee the moral implementation of emotion recognition technologies in practical settings.

8. Refernces:

1. Mollahosseini, A., Hasani, B., Salvador, M.J., Abdollahi, H., Chan, D. and Mahoor, M.H., 2016. Facial Expression Recognition from World Wild Web. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, pp.58-65.

2. Goodfellow, I., Bengio, Y., and Courville, A., 2016. *Deep Learning*. MIT Press.

3. Kahou, S.E., Bouthillier, X., Lamblin, P., Gülçehre, Ç., Michalski, V., Konda, K.R., Jean, S., Froumenty, P., Dauphin, Y.N., Boulanger-Lewandowski, N., Ferrari, R.C., Mirza, M., Warde-Farley, D., Courville, A.C., Vincent, P., Memisevic, R., Pal, C.J. and Bengio, Y., 2015. EmoNets: Multimodal deep learning approaches for emotion recognition in video. *Journal on Multimodal User Interfaces*, 10(2), pp.99-111. Available at: https://doi.org/10.1007/s12193-015-0195-2.

4.Tariq, U., Hussain, A., Zhang, Z., & Ahmad, S., 2019. Facial Expression Recognition by Transfer Learning for Small Datasets. *SpringerLink*. Available at: https://link.springer.com/article/10.1007/s12193-015-0195-2

5. Zhao, S., Cai, H., Liu, H., Zhang, J., and Chen, S., 2018. Feature Selection Mechanism in CNNs for Facial Expression Recognition. In *Proceedings of the British Machine Vision Conference (BMVC) 2018*. Available at: https://dblp.org/rec/conf/bmvc/ZhaoCLZC18

6. Sabour et al. (2017): Sabour, S., Frosst, N., and Hinton, G.E., 2017. Dynamic Routing Between Capsules. Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS 2017), pp. 3856-3866. Available at: https://arxiv.org/abs/1710.09829

7. Kang et al. (2020): Kang, G., Kim, Y., and Kwon, G., 2020. Real-time facial expression recognition system using CNNs. *IEEE Access*, 8, pp. 177680-177688. Available at: https://ieeexplore.ieee.org/document/9212368

8. Zeng, Z., Pantic, M., Roisman, G. I., & Huang, T. S., 2009. A survey of affect recognition methods: Audio, visual, and spontaneous expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(1), pp. 39-58. Available at: IEEE Xplore

9.Baltrusaitis, T., Zadeh, A., Lim, Y. C., & Morency, L. P., 2018. OpenFace 2.0: Facial behavior analysis toolkit. *Proceedings of the IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pp. 59-66. Available at: IEEE Xplore

10. Li, S., Deng, W., & Du, J., 2018. Reliable crowdsourcing and deep locality-preserving learning for unconstrained facial expression recognition. *IEEE Transactions on Image Processing*, 28(1), pp. 356-370. Available at: IEEE Xplore

11.Valstar, M. F., & Pantic, M., 2012. Fully automatic recognition of the temporal phases of facial actions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(1), pp. 28-43. Available at: IEEE Xplore

12.McDuff, D., El Kaliouby, R., & Picard, R. W., 2015. Crowdsourcing facial responses to online videos. *IEEE Transactions on Affective Computing*, 6(1), pp. 21-33. Available at: IEEE Xplore

13.Saffer, D. (2008). Designing gestural interfaces: Touchscreens and Interactive Devices. “O’Reilly Media, Inc.”

14.Pantic, M., & Bartlett, M. S., 2007. Machine understanding of facial expressions in face-to-face communication: A survey. Image and Vision Computing, 27(12), pp. 1743-1759.

15. He, K., Zhang, X., Ren, S., & Sun, J., 2016. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778.Available at: IEEE Xplore

16. LeCun, Y., Bengio, Y., & Hinton, G., 2015. Deep Learning. *Nature*, 521, pp. 436-444.Available at: Nature [Accessed 28 August 2024].

17. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779-788.Available at: IEEE Xplore

18. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*.Available at: arXiv

19. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C., 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510-4520.Available at: IEEE Xplore

20.Tzirakis, P., Trigeorgis, G., Nicolaou, M. A., Schuller, B. W., & Zafeiriou, S., 2017. End-to-End Multimodal Emotion Recognition using Deep Neural Networks. *IEEE Journal of Selected Topics in Signal Processing*, 11(8), pp. 1301-1309.Available at: IEEE Xplore

21. Huh, M., Agrawal, P., & Efros, A. A., 2016. What makes ImageNet good for transfer learning? *arXiv preprint arXiv:1608.08614*.Available at: arXiv

22. Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S., 2014. CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 806-813.Available at: IEEE Xplore

23. Sokolova, M., & Lapalme, G., 2009. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), pp. 427-437.Available at: ScienceDirect

24.Jeni, L. A., Cohn, J. F., & De La Torre, F., 2013. Facing Imbalanced Data—Recommendations for the Use of Performance Metrics. *Proceedings of the 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction (ACII)*, pp. 245-251.Available at: IEEE Xplore

25. Grandini, M., Bagli, E., & Visani, G., 2020. Metrics for Multi-Class Classification: an Overview. *arXiv preprint arXiv:2008.05756*.Available at: arXiv

26. Powers, D. M. W., 2011. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2(1), pp. 37-63.

27. Ko, B. C., 2018. A Brief Review of Facial Emotion Recognition Based on Visual Information. *Sensors*, 18(2), 401.

28. Yosinski, J., Clune, J., Bengio, Y., & Lipson, H., 2014. How Transferable Are Features in Deep Neural Networks? *Advances in Neural Information Processing Systems (NIPS 2014)*, pp. 3320-3328.

9. Appendices

Goggle Colab Link :

https://colab.research.google.com/drive/1nT-jToBW79Pv2wgPL_yTR0ZFSbQ-sXsS?usp=sharing

GitHub Link:

https://github.com/RamyasriManyala/HumanEmotionDetection

Code :

```
import os  # Handles file and directory operations.
import numpy as np  # Supports array operations and numerical computations.
import matplotlib.pyplot as plt  # Enables data visualization through plots and charts.
from sklearn.model_selection import train_test_split  # Splits data into training and testing sets.
from keras.models import Sequential  # Builds a linear stack of neural network layers.
from keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D  # Provides essential
layers for constructing a CNN.
from tensorflow.keras.preprocessing.image import ImageDataGenerator  # Performs real-time
data augmentation for images.
from keras.utils import to_categorical  # Converts class labels to one-hot encoded vectors.
from keras.preprocessing import image  # Loads and preprocesses images for model input.
import cv2  # Offers tools for image and video processing (OpenCV).
import pandas as pd  # Manages and analyzes structured data in DataFrames.
import seaborn as sns  # Creates advanced visualizations with a focus on statistical graphics.

# Import the drive module from google.colab to interact with Google Drive
from google.colab import drive

# Mount the Google Drive to the '/content/drive' directory
# This will prompt to authenticate and provide access to Google Drive files
drive.mount('/content/drive')

# Define the path to the training dataset directory.
data_dir_train = '/content/drive/MyDrive/Data/FinalProject/Dataset/train'

# Define the path to the testing dataset directory.
data_dir_test = '/content/drive/MyDrive/Data/FinalProject/Dataset/test'

# Define the list of emotion labels corresponding to different emotions.
emotion_labels = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

# Function to load images from the specified directory and their corresponding labels.
def load_images(data_dir, emotion_labels):
    images = []  # Initialize an empty list to store images.
    labels = []  # Initialize an empty list to store labels.

    # Loop through each label in the emotion_labels list.
    for label in emotion_labels:
        # Construct the path to the directory containing images for the current label.
        img_dir = os.path.join(data_dir, label)

        # Check if the directory exists; if not, print a message and continue to the next label.
        if not os.path.exists(img_dir):
            print(f"Directory {img_dir} does not exist")
            continue
```

```
        # Loop through each image file in the directory.
        for img_name in os.listdir(img_dir):
            img_path = os.path.join(img_dir, img_name)  # Construct the full path to the image.

            try:
                # Load the image with a target size of 48x48 pixels and grayscale mode.
                img = image.load_img(img_path, target_size=(48, 48), color_mode='grayscale')

                # Convert the loaded image to a NumPy array.
                img = image.img_to_array(img)

                # Convert the grayscale image to RGB by repeating the grayscale values across the
three color channels.
                img = np.repeat(img, 3, axis=-1)

                # Normalize the pixel values to the range [0, 1].
                img = img / 255.0

                # Append the processed image to the images list.
                images.append(img)

                # Append the corresponding label index to the labels list.
                labels.append(emotion_labels.index(label))

            # Handle exceptions during image loading and print the error message.
            except Exception as e:
                print(f"Error loading image {img_path}: {e}")

    # Return the images and labels as NumPy arrays.
    return np.array(images), np.array(labels)

# Load the training dataset images and their labels.
X_train, y_train = load_images(data_dir_train, emotion_labels)

# Load the testing dataset images and their labels.
X_test, y_test = load_images(data_dir_test, emotion_labels)

# Ensure labels are one-hot encoded for use in training models.
y_train = to_categorical(y_train, num_classes=len(emotion_labels))
y_test = to_categorical(y_test, num_classes=len(emotion_labels))

import pandas as pd  # Import the pandas library for data manipulation.

# Create a DataFrame from the labels, mapping one-hot encoded labels back to their original
emotion names.
labels_df = pd.DataFrame({'label': [emotion_labels[label] for label in np.argmax(y_train, axis=1)]})

# Set up the plot with a specified figure size.
plt.figure(figsize=(10, 6))

# Create a count plot to visualize the distribution of classes in the training data.
# 'order=emotion_labels' ensures that the x-axis labels are displayed in the correct order.
sns.countplot(x='label', data=labels_df, order=emotion_labels)

# Add a title to the plot.
plt.title('Distribution of Classes in Training Data')

# Label the x-axis.
plt.xlabel('Class')
```

```
# Label the y-axis.
plt.ylabel('Number of Images')

# Ensure the x-axis labels are properly aligned with the class labels.
plt.xticks(ticks=np.arange(len(emotion_labels)), labels=emotion_labels)

# Display the plot.
plt.show()

def display_sample_images(X, y, emotion_labels, num_samples=5):
    """
    Display a grid of sample images for each emotion label.

    Parameters:
    X : numpy array
        Array containing the image data.
    y : numpy array
        Array containing the one-hot encoded labels.
    emotion_labels : list
        List of emotion labels corresponding to the classes.
    num_samples : int, optional
        Number of sample images to display for each emotion (default is 5).
    """
    plt.figure(figsize=(15, 10))  # Set the size of the figure.

    # Loop through each emotion label.
    for i, label in enumerate(emotion_labels):
        # Find the indices of images corresponding to the current label.
        label_indices = np.where(np.argmax(y, axis=1) == i)[0]

        # Randomly select a specified number of images from the current label.
        sample_indices = np.random.choice(label_indices, num_samples, replace=False)

        # Loop through the selected images and display them.
        for j, idx in enumerate(sample_indices):
            # Create a subplot for each image.
            plt.subplot(len(emotion_labels), num_samples, i*num_samples + j + 1)

            # Convert the RGB image to grayscale by averaging the color channels.
            gray_image = np.mean(X[idx], axis=-1)

            # Display the grayscale image.
            plt.imshow(gray_image, cmap='gray')

            # Remove the axis for a cleaner display.
            plt.axis('off')

            # Add a title with the label name to the first image in each row.
            if j == 0:
                plt.title(label)

    # Display the complete grid of images.
    plt.show()

# Call the function to display sample images from the training dataset.
display_sample_images(X_train, y_train, emotion_labels)

print(f"Image shape: {X_train[0].shape}”)

# Calculate the mean pixel value across all images in the training dataset.
```

```
mean_pixel_value = np.mean(X_train)

# Calculate the standard deviation of pixel values across all images in the training dataset.
std_pixel_value = np.std(X_train)

# Print out the calculated mean pixel value.
print(f"Mean pixel value: {mean_pixel_value}")

# Print out the calculated standard deviation of pixel values.
print(f"Standard deviation of pixel values: {std_pixel_value}")

# Set up the figure size for the plot.
plt.figure(figsize=(10, 6))

# Create a histogram of pixel intensities for the first image in the training dataset.
# 'flatten()' converts the 2D image array into a 1D array of pixel values.
plt.hist(X_train[0].flatten(), bins=50, color='gray')

# Add a title to the histogram.
plt.title('Pixel Intensity Distribution of a Sample Image')

# Label the x-axis to indicate it represents pixel intensity values.
plt.xlabel('Pixel Intensity')

# Label the y-axis to indicate it represents the frequency of each intensity value.
plt.ylabel('Frequency')

# Display the histogram.
plt.show()

from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Define the data augmentation generator with various transformations.
datagen = ImageDataGenerator(
    rotation_range=20,          # Rotate images randomly by up to 20 degrees.
    width_shift_range=0.2,      # Shift images horizontally by up to 20% of the width.
    height_shift_range=0.2,     # Shift images vertically by up to 20% of the height.
    shear_range=0.2,            # Apply random shearing transformations.
    zoom_range=0.2,             # Apply random zooming within the specified range.
    horizontal_flip=True,       # Randomly flip images horizontally.
    fill_mode='nearest'         # Fill in pixels that may have been lost after a transformation.
)

# Prepare a single image from the training set for augmentation by reshaping it for the generator.
# The image is reshaped to (1, 48, 48, 3) to indicate a single RGB image with 48x48 pixels.
sample_image = X_train[0].reshape(1, 48, 48, 3)

# Create an iterator that will generate augmented images from the sample image.
aug_iter = datagen.flow(sample_image)

# Set up a figure for displaying the augmented images.
plt.figure(figsize=(12, 12))

# Generate and display 9 augmented images.
for i in range(9):
    plt.subplot(3, 3, i+1)  # Arrange the images in a 3x3 grid.
    batch = next(aug_iter)  # Generate the next batch of augmented images.
    image_augmented = batch[0]  # Extract the first (and only) image from the batch.
    plt.imshow(image_augmented)  # Display the augmented image.
```

```
    plt.axis('off')  # Turn off axis labels for a cleaner look.

# Show the figure with the augmented images.
plt.show()

# Select a few images from the training set and convert each to grayscale and flatten.
# The flattening process converts the 2D image arrays into 1D arrays of pixel values.
images = [np.mean(X_train[i], axis=-1).flatten() for i in range(4)]

# Calculate the correlation matrix between these flattened grayscale images.
# The correlation matrix will show how similar the pixel intensity patterns are between the
selected images.
correlation_matrix = np.corrcoef(images)

# Set up the figure size for the correlation matrix heatmap.
plt.figure(figsize=(10, 6))

# Plot the correlation matrix as a heatmap using seaborn.
# 'annot=True' displays the correlation coefficients on the heatmap.
# 'cmap="coolwarm"' sets the color scheme, with 'cool' colors for negative correlations and
'warm' colors for positive correlations.
# 'xticklabels' and 'yticklabels' label the axes with the corresponding image numbers.
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
         xticklabels=['Image 1', 'Image 2', 'Image 3', 'Image 4'],
         yticklabels=['Image 1', 'Image 2', 'Image 3', 'Image 4'])

# Add a title to the heatmap.
plt.title('Correlation Matrix of Pixel Values Between Images')

# Display the heatmap.
plt.show()

# Convert one-hot encoded labels back to categorical form for both training and testing data.
# This is done by finding the index of the maximum value along the second axis (axis=1) for each
label.
y_train_labels = np.argmax(y_train, axis=1)
y_test_labels = np.argmax(y_test, axis=1)

# Create DataFrames to hold the categorical labels for easier visualization.
labels_df_train = pd.DataFrame({'label': y_train_labels})
labels_df_test = pd.DataFrame({'label': y_test_labels})

# Plot the distribution of classes in the training data.
plt.figure(figsize=(10, 6))  # Set up the figure size.
sns.countplot(x='label', data=labels_df_train, palette='viridis')  # Create a count plot with a 'viridis'
color palette.
plt.title('Distribution of Classes in Training Data')  # Add a title to the plot.
plt.xlabel('Class')  # Label the x-axis.
plt.ylabel('Number of Images')  # Label the y-axis.
plt.xticks(ticks=np.arange(len(emotion_labels)), labels=emotion_labels)  # Set the x-axis tick labels
to the emotion labels.
plt.show()  # Display the plot.

# Plot the distribution of classes in the testing data.
plt.figure(figsize=(10, 6))  # Set up the figure size.
sns.countplot(x='label', data=labels_df_test, palette='magma')  # Create a count plot with a
'magma' color palette.
plt.title('Distribution of Classes in Testing Data')  # Add a title to the plot.
plt.xlabel('Class')  # Label the x-axis.
plt.ylabel('Number of Images')  # Label the y-axis.
```

```
plt.xticks(ticks=np.arange(len(emotion_labels)), labels=emotion_labels)  # Set the x-axis tick labels to the emotion labels.
plt.show()  # Display the plot.

from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# Flatten the images in the training dataset.
# This reshapes each image from 3D (height, width, channels) to 1D (a single vector of pixel values).
X_train_flat = X_train.reshape(X_train.shape[0], -1)

# Sample a smaller subset of the training data, specifically 1% of the dataset.
sample_size = int(0.01 * X_train.shape[0])  # Calculate 1% of the total number of images.
indices = np.random.choice(X_train.shape[0], sample_size, replace=False)  # Randomly select sample_size indices without replacement.
X_train_sample = X_train_flat[indices]  # Select the images corresponding to the sampled indices.
y_train_sample = y_train_labels[indices]  # Select the labels corresponding to the sampled indices.

# Apply Principal Component Analysis (PCA) to reduce the dimensionality of the sampled data to 2 components.
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_sample)

# Plot the results of the PCA.
plt.figure(figsize=(10, 6))  # Set up the figure size.

# Loop through each emotion label and plot the corresponding points in the PCA-transformed space.
for i, label in enumerate(emotion_labels):
    indices = np.where(y_train_sample == i)[0]  # Find the indices of the samples belonging to the current label.
    plt.scatter(X_pca[indices, 0], X_pca[indices, 1], label=label, alpha=0.5)  # Plot the points with some transparency (alpha=0.5).

# Add a title to the plot.
plt.title('PCA of Training Data (1% Sample)')

# Label the x-axis and y-axis to indicate the principal components.
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

# Add a legend to the plot to identify the emotion labels.
plt.legend()

# Display the plot.
plt.show()

import cv2  # Import the OpenCV library for image processing.

def calculate_average_image(images):
    """
    Calculate the average image from a set of images.

    Parameters:
    images : numpy array
        Array containing the images.
```

```
    Returns:
    numpy array
        The average image.
    """
    return np.mean(images, axis=0)  # Compute the mean of the images along the first axis.

# Set up the figure for displaying the average images of each class.
plt.figure(figsize=(15, 5))

# Loop through each emotion label.
for i, label in enumerate(emotion_labels):
    # Find the indices of the images corresponding to the current label.
    indices = np.where(y_train_labels == i)[0]

    # Skip this label if there are no images associated with it.
    if len(indices) == 0:
        print(f"Skipping class '{label}' due to no images.")
        continue

    # Calculate the average image for the current label.
    avg_image = calculate_average_image(X_train[indices])

    # Convert the average image to grayscale by averaging the color channels.
    avg_image_gray = np.mean(avg_image, axis=-1)

    # Normalize the grayscale image to the range [0, 1].
    avg_image_gray = (avg_image_gray - np.min(avg_image_gray)) / (np.max(avg_image_gray) -
np.min(avg_image_gray))

    # Enhance the contrast of the image using contrast stretching with OpenCV's normalize
function.
    avg_image_gray = cv2.normalize(avg_image_gray, None, alpha=0, beta=1,
norm_type=cv2.NORM_MINMAX)

    # Plot the average image in the current subplot.
    plt.subplot(1, len(emotion_labels), i + 1)
    plt.imshow(avg_image_gray, cmap='gray')  # Display the image in grayscale.
    plt.axis('off')  # Turn off axis labels for a cleaner look.
    plt.title(label)  # Add the label as the title of the subplot.

# Add a title for the entire figure.
plt.suptitle("Average Image for Each Class")

# Display the figure with all the average images.
plt.show()

from sklearn.utils import class_weight  # Import the class_weight module from sklearn to handle
class imbalance.

# Calculate class weights to handle imbalanced classes in the dataset.
# 'class_weight="balanced"' ensures that the weights are inversely proportional to the class
frequencies.
# 'classes=np.unique(np.argmax(y_train, axis=1))' provides the unique class labels in the training
data.
# 'y=np.argmax(y_train, axis=1)' converts the one-hot encoded labels back to categorical form.
class_weights = class_weight.compute_class_weight(class_weight='balanced',
                                                  classes=np.unique(np.argmax(y_train, axis=1)),
                                                  y=np.argmax(y_train, axis=1))
```

```
# Convert the class weights array to a dictionary where the keys are class labels and values are
the corresponding weights.
class_weights = dict(enumerate(class_weights))

from tensorflow.keras.applications import VGG16  # Import the VGG16 model from Keras
applications.
from tensorflow.keras.optimizers import Adam  # Import the Adam optimizer from Keras.

# Load the VGG16 model pre-trained on the ImageNet dataset, excluding the fully connected
layers at the top.
# 'include_top=False' means the model is loaded without the top fully connected layers, allowing
us to add our own.
# 'input_shape=(48, 48, 3)' specifies the input size for the model, matching the dimensions of our
dataset.
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(48, 48, 3))

# Fine-tune only the last few layers of the VGG16 model.
# Loop through the layers of VGG16, setting the 'trainable' attribute to False for all but the last
four layers.
for layer in vgg16.layers[:-4]:  # Only the last four layers will remain trainable.
    layer.trainable = False  # Freeze the layers, so their weights are not updated during training.

from keras.callbacks import ReduceLROnPlateau, EarlyStopping  # Import callbacks for learning
rate adjustment and early stopping.
from tensorflow.keras.applications import VGG16  # Import the VGG16 model from Keras
applications.
from tensorflow.keras.optimizers import Adam  # Import the Adam optimizer from Keras.

# Load the VGG16 model pre-trained on ImageNet without the top fully connected layers.
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(48, 48, 3))

# Fine-tune the last few layers of the VGG16 model.
# Freeze all layers except for the last four by setting 'trainable' to False.
for layer in vgg16.layers[:-4]:
    layer.trainable = False

# Create a Sequential model and add the VGG16 base model.
model = Sequential()
model.add(vgg16)

# Add a Flatten layer to convert the 3D feature maps to 1D feature vectors.
model.add(Flatten())

# Add a fully connected Dense layer with 256 units and ReLU activation.
model.add(Dense(256, activation='relu'))

# Add a Dropout layer with a dropout rate of 0.5 to prevent overfitting.
model.add(Dropout(0.5))

# Add a final Dense layer with the number of units equal to the number of emotion classes and
softmax activation for multi-class classification.
model.add(Dense(len(emotion_labels), activation='softmax'))

# Compile the model with the Adam optimizer, using a low learning rate, categorical cross-entropy
loss, and accuracy as a metric.
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Set up data augmentation using ImageDataGenerator to artificially increase the diversity of the
training data.
```

```
datagen = ImageDataGenerator(
    rotation_range=30,           # Randomly rotate images by up to 30 degrees.
    zoom_range=0.2,              # Randomly zoom into images by up to 20%.
    width_shift_range=0.1,       # Randomly shift images horizontally by up to 10% of the width.
    height_shift_range=0.1,      # Randomly shift images vertically by up to 10% of the height.
    horizontal_flip=True,        # Randomly flip images horizontally.
    fill_mode='nearest'          # Fill any missing pixels after transformations using the nearest pixels.
)

# Create generators for training and validation data with a batch size of 64.
train_generator = datagen.flow(X_train, y_train, batch_size=64)
validation_generator = datagen.flow(X_test, y_test, batch_size=64)

# Set up callbacks for training:
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.00001)
# Reduce the learning rate by a factor of 0.2 if the validation loss doesn't improve for 5 epochs,
with a minimum learning rate of 0.00001.

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
# Stop training early if the validation loss doesn't improve for 10 epochs, and restore the model
weights from the best epoch.

# Train the model using the training generator and validate using the validation generator.
# The training process will run for up to 50 epochs, with callbacks for learning rate reduction and
early stopping.
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=50,
    callbacks=[reduce_lr, early_stopping]
)

# Save the model
model.save('emotion_detection_model_vgg16.h5')

# Evaluate the model on the validation data using the validation generator.
# This returns the loss and accuracy metrics on the validation set.
loss, accuracy = model.evaluate(validation_generator)

# Print the loss value to see how well the model is performing on the validation set.
print(f'Loss: {loss}')

# Print the accuracy value to see the percentage of correctly classified images on the validation
set.
print(f'Accuracy: {accuracy}')

# Plot the training and validation accuracy and loss over the epochs.

# Set up the figure with two subplots, side by side.
plt.figure(figsize=(12, 4))

# Plot the training and validation accuracy on the first subplot.
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')  # Plot training accuracy.
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')  # Plot validation accuracy.
plt.legend()  # Display a legend to distinguish between training and validation accuracy.
plt.title('Accuracy')  # Add a title to the accuracy plot.

# Plot the training and validation loss on the second subplot.
plt.subplot(1, 2, 2)
```

```
plt.plot(history.history['loss'], label='Train Loss')  # Plot training loss.
plt.plot(history.history['val_loss'], label='Validation Loss')  # Plot validation loss.
plt.legend()  # Display a legend to distinguish between training and validation loss.
plt.title('Loss')  # Add a title to the loss plot.

# Display the plots.
plt.show()

# Use the trained model to make predictions on the test data.
predictions = model.predict(X_test)

# Convert the predicted probabilities to class labels by taking the index of the maximum value in
each prediction.
predicted_classes = np.argmax(predictions, axis=1)

# Convert the true one-hot encoded labels of the test data back to categorical form by taking the
index of the maximum value in each label.
true_classes = np.argmax(y_test, axis=1)

from sklearn.metrics import confusion_matrix  # Import the confusion_matrix function from
sklearn.
import seaborn as sns  # Import seaborn for easier visualization of the confusion matrix.
import matplotlib.pyplot as plt  # Import matplotlib for plotting.

# Calculate the confusion matrix to compare the predicted classes with the true classes.
cm = confusion_matrix(true_classes, predicted_classes)

# Set up the figure for displaying the confusion matrix.
plt.figure(figsize=(10, 8))

# Plot the confusion matrix as a heatmap.
# 'annot=True' adds the count annotations on the heatmap.
# 'fmt="d"' ensures that the annotations are displayed as integers.
# 'cmap="Blues"' sets the color scheme for the heatmap.
# 'xticklabels' and 'yticklabels' set the labels on the x and y axes to the emotion labels.
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=emotion_labels,
yticklabels=emotion_labels)

# Label the x-axis as 'Predicted'.
plt.xlabel('Predicted')

# Label the y-axis as 'Actual'.
plt.ylabel('Actual')

# Add a title to the confusion matrix.
plt.title('Confusion Matrix')

# Display the plot.
plt.show()

from sklearn.metrics import precision_score, recall_score, f1_score  # Import metrics for model
evaluation.

# Calculate the precision for each class.
# 'average=None' returns the precision for each class separately rather than averaging them.
precision = precision_score(true_classes, predicted_classes, average=None)

# Calculate the recall for each class.
recall = recall_score(true_classes, predicted_classes, average=None)
```

```
# Calculate the F1-score for each class.
f1 = f1_score(true_classes, predicted_classes, average=None)

# Print the precision, recall, and F1-score for each emotion class.
for idx, label in enumerate(emotion_labels):
    print(f"{label} - Precision: {precision[idx]:.2f}, Recall: {recall[idx]:.2f}, F1-Score: {f1[idx]:.2f}")
    # .2f formats the values to two decimal places for readability.

import numpy as np
import matplotlib.pyplot as plt

def plot_predictions(images, true_labels, predicted_labels, emotion_labels,
num_images_per_class=3):
    """
    Plots a grid of images with their true and predicted labels, displaying num_images_per_class
images per emotion.

    Parameters:
    - images: Array of images.
    - true_labels: Array of true labels corresponding to the images.
    - predicted_labels: Array of predicted labels corresponding to the images.
    - emotion_labels: List of emotion label names.
    - num_images_per_class: Number of images to display per emotion class.
    """

    # Initialize the plot
    plt.figure(figsize=(15, len(emotion_labels) * 2.5))

    for i, label in enumerate(emotion_labels):
        # Find indices of images belonging to the current emotion label
        indices = np.where((true_labels == i) & (predicted_labels == i))[0]
        if len(indices) >= num_images_per_class:
            selected_indices = np.random.choice(indices, num_images_per_class, replace=False)
        else:
            selected_indices = indices  # Use all available images if fewer than required

        for j, idx in enumerate(selected_indices):
            plt.subplot(len(emotion_labels), num_images_per_class, i * num_images_per_class + j + 1)
            plt.imshow(images[idx].reshape(48, 48, 3))
            plt.title(f"True: {label}\nPred: {emotion_labels[predicted_labels[idx]]}")
            plt.axis('off')

    plt.tight_layout()
    plt.show()

# Assuming the following variables are already defined:
# - X_test: Test images.
# - true_classes: True labels for the test images.
# - predicted_classes: Predicted labels from the model.
# - emotion_labels: List of emotion label names.

plot_predictions(X_test, true_classes, predicted_classes, emotion_labels,
num_images_per_class=3)

# Save the entire model (architecture + weights)
model.save(‘emotion_detection_model_vgg16_full.h5')

from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
```

```
from PIL import Image as PILImage
import numpy as np
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import os

def take_photo(filename='photo.jpg', quality=0.8):
    js = Javascript('''
        async function takePhoto(quality) {
            const div = document.createElement('div');
            const capture = document.createElement('button');
            capture.textContent = 'Capture';
            div.appendChild(capture);

            const video = document.createElement('video');
            video.style.display = 'block';
            const stream = await navigator.mediaDevices.getUserMedia({video: true});

            document.body.appendChild(div);
            div.appendChild(video);
            video.srcObject = stream;
            await video.play();

            google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

            await new Promise((resolve) => capture.onclick = resolve);

            const canvas = document.createElement('canvas');
            canvas.width = video.videoWidth;
            canvas.height = video.videoHeight;
            canvas.getContext('2d').drawImage(video, 0, 0);
            stream.getVideoTracks()[0].stop();
            div.remove();
            return canvas.toDataURL('image/jpeg', quality);
        }
    ''')
    display(js)
    data = eval_js('takePhoto({})'.format(quality))
    binary = b64decode(data.split(',')[1])
    with open(filename, 'wb') as f:
        f.write(binary)
    return os.path.abspath(filename)

try:
    # Capture an image from the webcam using the take_photo function.
    filename = take_photo()
    print('Saved to {}'.format(filename))  # Print the filename where the image is saved.

    # Display the captured image.
    display(PILImage.open(filename))

    # Load the pre-trained emotion detection model.
    model = load_model('emotion_detection_model_vgg16_full.h5')

    # Define the emotion labels corresponding to the model's output classes.
    emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

    # Preprocess the captured image for model prediction.
    image = PILImage.open(filename)  # Open the saved image.
    image = image.convert('RGB')  # Ensure the image is in RGB format.
```

```
    image = image.resize((48, 48))  # Resize the image to match the input size expected by the
model.
    image = img_to_array(image)  # Convert the image to a NumPy array.
    image = np.expand_dims(image, axis=0)  # Add an extra dimension to match the model's input
shape (batch size, height, width, channels).
    image = image / 255.0  # Normalize the image pixel values to the range [0, 1].

    # Predict the emotion using the pre-trained model.
    prediction = model.predict(image)

    # Determine the emotion label with the highest prediction probability.
    emotion = emotion_labels[np.argmax(prediction)]

    # Print the detected emotion.
    print(f'Detected Emotion: {emotion}')

except Exception as err:
    # Handle any errors that occur during the process and print the error message.
    print(str(err))

from tensorflow.keras.applications import VGG16
from keras.layers import Input, Flatten, Dense, Dropout
from keras.models import Model
from tensorflow.keras.optimizers import Adam

# Clear the session to avoid any contamination
from keras import backend as K
K.clear_session()

# Define the input shape
input_shape = (48, 48, 3)

# Input layer
input_layer = Input(shape=input_shape)

# Load the VGG16 model without the top classification layer
vgg16_base = VGG16(weights='imagenet', include_top=False, input_tensor=input_layer)

# Fine-tune the last few layers
for layer in vgg16_base.layers[:-4]:
    layer.trainable = False

# Add custom layers on top of the VGG16 base
x = Flatten()(vgg16_base.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
output_layer = Dense(7, activation='softmax')(x)  # Assuming 7 emotion classes

# Create the model
model = Model(inputs=input_layer, outputs=output_layer)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Print the model summary
try:
    model.summary()
except ValueError as e:
    print("Error during model summary:", e)
```

```
from keras.models import load_model  # Import the function to load a pre-trained Keras model.
from keras.preprocessing.image import img_to_array  # Import the function to convert an image
to a NumPy array.
from PIL import Image as PILImage  # Import the Image class from the PIL library for image
processing.
import numpy as np  # Import NumPy for numerical operations.

# Load the pre-trained emotion detection model.
try:
    model = load_model('emotion_detection_model_vgg16_full.h5')  # Load the model from the
specified file.
    model.summary()  # Print the model summary to verify that the model has been loaded
correctly.
except Exception as e:
    print("Error loading model:", e)  # Print an error message if the model fails to load.

# Define the emotion labels corresponding to the output classes of the model.
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

# Load and preprocess the image for prediction.
image_path = '/content/photo.jpg'  # Specify the path to the image file.
image = PILImage.open(image_path)  # Open the image using PIL.
image = image.convert('RGB')  # Convert the image to RGB format.
image = image.resize((48, 48))  # Resize the image to match the input size expected by the model
(48x48 pixels).
image = img_to_array(image)  # Convert the image to a NumPy array.
image = np.expand_dims(image, axis=0)  # Add an extra dimension to match the model's input
shape (batch size, height, width, channels).
image = image / 255.0  # Normalize the image pixel values to the range [0, 1].

# Predict the emotion using the pre-trained model.
try:
    prediction = model.predict(image)  # Perform the prediction.
    emotion = emotion_labels[np.argmax(prediction)]  # Determine the emotion label with the
highest prediction probability.
    print(f'Detected Emotion: {emotion}')  # Print the detected emotion.
except Exception as e:
    print("Error during prediction:", e)  # Print an error message if there is an issue during
prediction.
```