NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL

Project Report on

Facial Expression Recognition Using Machine Learning

By: Shivani Saran
Roll No: 242CD022
Branch: CDS, 1st Semester, M.Tech
Course: MA722 Foundation of Machine Learning Algorithms
Guided By: A. Senthil Thilak

Table of Contents

1. **Introduction** 2
2. **Motivation** 2
3. **Preliminary Requirements** 2
4. **Literature Survey** 2
5. **Existing Methods** 3
6. **Justification of Chosen Method** 4
7. **Methodology** 5
   - 7.1 Dataset Preparation 5
   - 7.2 Data Preprocessing 6
   - 7.3 Label Encoding 7
   - 7.4 Building the CNN Model 8
   - 7.5 Real Time Emotion Detection 13
8. **Result and discussion** 14
9. **Conclusion** 15
10. **Applications** 15
11. **Limitations** 16
12. **References** 16

1. Introduction

Facial Expression Recognition (FER) is the task of identifying human emotions based on facial expressions. It has significant applications in areas such as human-computer interaction, psychological studies, robotics, and security systems. In this project, a machine learning model is developed to classify facial expressions into seven categories: angry, disgust, fear, happy, neutral, sad, and surprise. The model utilizes Convolutional Neural Networks (CNNs) for feature extraction and classification.

2. Motivation

The motivation for this project arises from the importance of understanding human emotions, which can greatly enhance interaction between humans and machines. The ability of machines to detect emotions accurately will open doors for advancements in various sectors, including healthcare, automotive safety systems, and entertainment.

3. Preliminary Requirements

To understand and execute this project, the following concepts and tools are required:

- **Machine Learning**: Basic understanding of supervised learning and neural networks.
- **Convolutional Neural Networks (CNNs)**: A deep learning technique for image classification.
- **Python Programming**: Knowledge of Python for data processing and model building.
- **Keras and TensorFlow**: Libraries used for implementing the machine learning model.
- **OpenCV**: Used for real-time video processing and face detection.

4. Literature Survey

Facial Expression Recognition (FER) has been an area of active research for decades, with techniques evolving from traditional feature-based methods to advanced deep learning approaches.

- **Traditional Methods:**
  Early approaches in FER relied heavily on manually crafted features. For instance:
  - **Local Binary Patterns (LBP):** These features encode local texture patterns of the face by comparing pixel intensities within small neighborhoods. LBP has been widely used due to its simplicity and computational efficiency.
  - **Histogram of Oriented Gradients (HOG):** HOG captures edge or gradient structures of facial features and is robust to illumination changes.

After extracting features, classifiers like **Support Vector Machines (SVM)** or **k-Nearest Neighbors (k-NN)** were used to categorize facial expressions into predefined classes. However, these methods suffered from limited robustness when handling real-world challenges like lighting variations, occlusion, and pose changes.

- **Deep Learning Models:**
  The advent of **Convolutional Neural Networks (CNNs)** revolutionized FER by automating feature extraction and improving classification accuracy. CNNs learn hierarchical representations directly from pixel-level data, identifying both low-level (edges, corners) and high-level (eyes, mouth) features. Unlike traditional methods, CNNs adaptively learn to recognize expressions even under challenging conditions like occlusion or variations in pose.
  Several studies highlight the effectiveness of CNNs when trained on large, annotated datasets.

- **Datasets:**
  Datasets play a critical role in FER research, providing diverse facial expression samples for training and testing. Notable datasets include:
  - **FER2013:** A widely used dataset comprising grayscale facial images labeled with seven expression classes (anger, disgust, fear, happiness, sadness, surprise, and neutral). Its large size and varied samples make it a standard benchmark for FER models.
  - Other datasets like **CK+ (Cohn-Kanade+), JAFFE, and AffectNet** also offer rich annotations, but FER2013 remains popular for CNN-based methods due to its scale and diversity.

5. Existing Methods

Over time, FER methods have transitioned from traditional techniques to deep learning-based approaches:

- **Traditional Methods:**
  These approaches depend on feature descriptors like LBP, HOG, and Gabor filters, combined with classical classifiers such as SVM or decision trees. While computationally efficient, they struggled with real-world challenges:
  - **Lighting Variability:** Substantial differences in lighting conditions affect the intensity of pixel values, making hand-crafted features less reliable.
  - **Pose Variations:** Facial orientations other than frontal pose degrade performance.
  - **Occlusions:** Objects like glasses, hands, or masks obscure key facial features.

- **Deep Learning Models:**
  CNNs and their variants have become the state-of-the-art for FER, surpassing traditional methods in performance and robustness. These models are capable of:
  - Learning from raw image data without explicit feature engineering.
  - Handling variations in facial expressions, lighting, and pose due to their large learning capacity and non-linearity.
- **Challenges in FER:**

  Despite advancements, FER still faces the following obstacles:

  1. **Individual Differences:** Variability in expression intensity and style across individuals.
  2. **Noise and Occlusion:** Presence of elements like glasses, masks, or headwear that partially block facial features.
  3. **Inconsistent Lighting:** Illumination changes can distort pixel-level information, impacting model predictions.

6. Justification of Chosen Method

The decision to use CNNs for this project stems from their proven ability to address the unique challenges of FER:

- **Automatic Feature Extraction:**
  Traditional methods rely on fixed feature extraction techniques, which may overlook critical information. CNNs, however, automatically extract the most relevant features for FER, enabling them to adapt to complex facial patterns and variations. For example, CNNs can identify subtle changes in facial muscle movements that correspond to specific expressions.
- **Accuracy:**
  Extensive studies demonstrate that CNNs consistently outperform traditional methods in FER benchmarks. By leveraging deeper architectures, techniques like transfer learning, and advanced optimizers, CNNs achieve higher accuracy and generalization on unseen data.
- **Scalability:**
  CNN-based models are highly scalable and can be extended to handle additional expression classes or integrate multimodal data (e.g., audio or text) for emotion analysis. Furthermore, as the amount of training data increases, CNNs can capitalize on their hierarchical structure to improve performance without requiring significant modifications.

7. Methodology

Step 1: Dataset Preparation

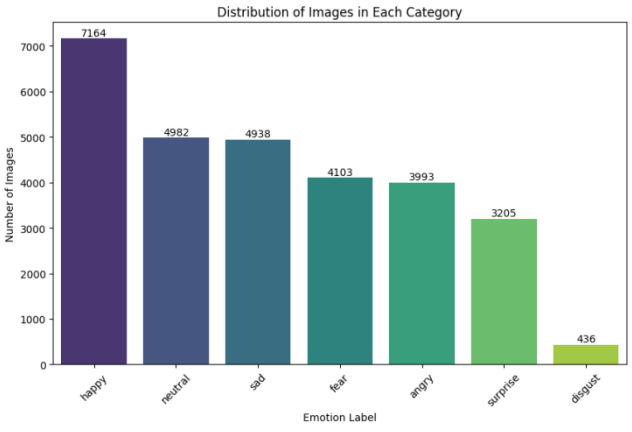
The dataset consists of labeled facial images stored in directories representing different emotion categories. The images are loaded and split into training and testing datasets.

```
Untitled-1

import os
import pandas as pd

TRAIN_DIR = 'images/train'
TEST_DIR = 'images/test'

def createdataframe(dir):
    image_paths = []
    labels = []
    for label in os.listdir(dir):
        for imagename in os.listdir(os.path.join(dir, label)):
            image_paths.append(os.path.join(dir, label, imagename))
            labels.append(label)
    return image_paths, labels
```



The dataset preparation step is crucial for organizing and structuring the data needed for training and testing the model. The createdataframe() function accomplishes this by traversing the specified directory, which contains subdirectories named after different emotion categories (e.g., happy, sad). Each subdirectory holds images that represent that specific emotion. The function reads each subdirectory, constructs the full path for each image, and collects these paths along with their associated labels. This process results in two lists: one for the image paths and another for the corresponding emotion labels. These lists serve as the foundation for loading the images and linking them with their correct labels, making them ready for input into the model for supervised learning. This structured approach ensures that each image can be accurately used for training or testing according to its assigned label.

Step 2: Data Preprocessing

- **Image Resizing**: Each image is resized to 48x48 pixels.
- **Grayscale Conversion**: Images are converted to grayscale to reduce complexity.
- **Normalization**: Pixel values are scaled to the range [0, 1].

Data processing

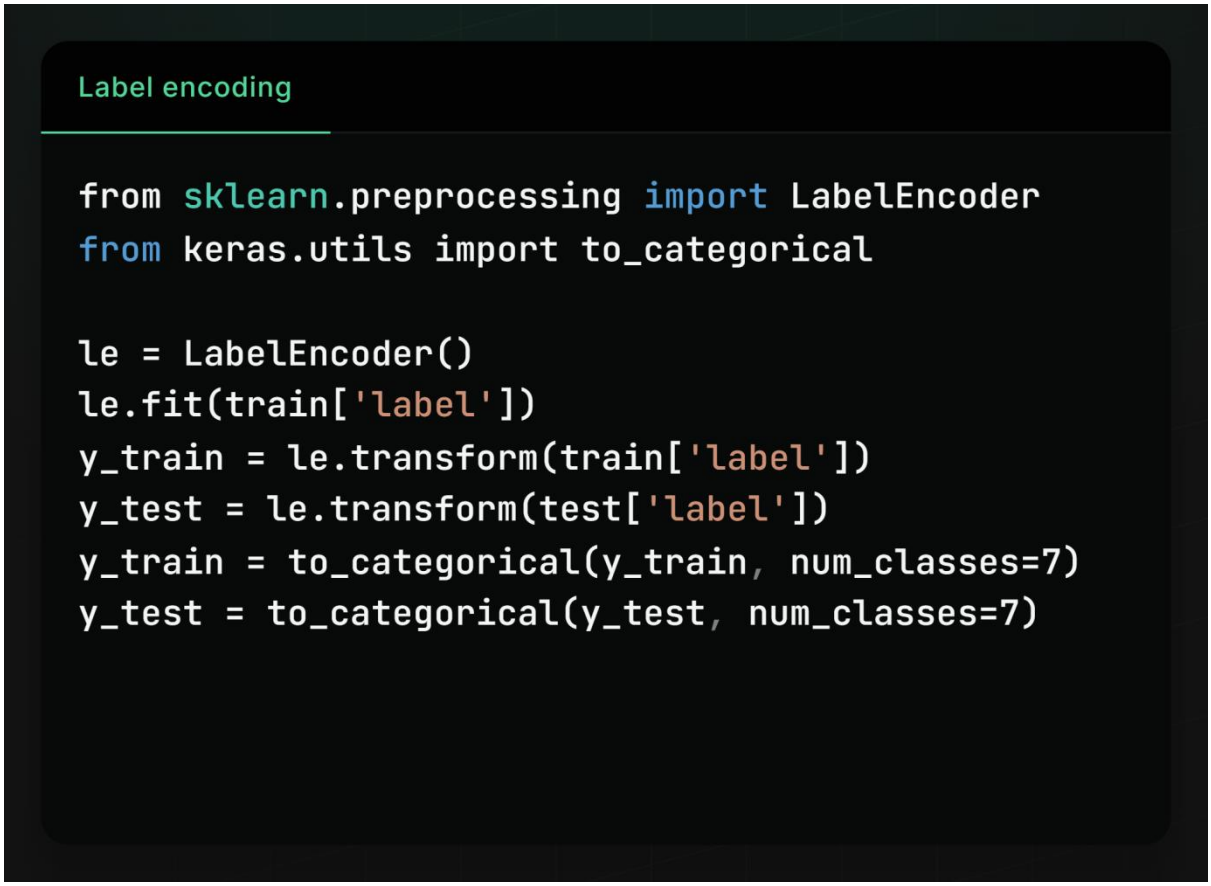
```
from keras.preprocessing.image import load_img
import numpy as np

def extract_features(images):
    features = []
    for image in images:
        img = load_img(image, grayscale=True)
        img = np.array(img)
        features.append(img)
    return np.array(features).reshape(len(features), 48, 48, 1) / 255.0
```

The data preprocessing step is essential to prepare the images for efficient training. First, each image is resized to a standard dimension of 48x48 pixels to ensure uniformity, which simplifies the model's input requirements and reduces computational overhead. Next, the images are converted to grayscale to focus on the essential features related to emotion, removing color information and thus reducing the complexity of the data. Finally, normalization is applied, scaling pixel values to the range [0, 1]. This step improves model performance by standardizing the input, which helps the model converge more effectively during training. The extract_features() function implements these steps by loading each image, converting it to grayscale, resizing it, and normalizing the pixel values to prepare the data for model input.

Step 3: Label Encoding

The emotion labels are encoded into numerical format using **LabelEncoder** from Scikit-learn.



Label encoding is used to transform categorical labels (such as emotions) into a numerical format that the model can process. This step ensures that each emotion category is represented as a distinct integer. Using LabelEncoder from Scikit-learn, the fit() method is applied to learn the mapping from the emotion labels in the training data. The transform() method then converts these labels into integers. For example, if the emotions are 'happy', 'sad', and 'angry', the encoder might map them to 0, 1, and 2, respectively.

After encoding, the labels are one-hot encoded using Keras' to_categorical() to create binary matrices. This encoding ensures that each label is represented as a vector, with a '1' at the index corresponding to the category and '0' elsewhere. This format is essential for multi-class classification tasks, as it helps the model learn to distinguish between different emotion classes. The resulting arrays y_train and y_test are ready for use in training and evaluation.

Step 4: Building the CNN Model

Overview of CNNs:

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing grid-like data, such as images. They are widely used for image classification tasks because they are excellent at automatically detecting patterns in images, such as edges, textures, and higher-level features. CNNs are made up of layers that perform specific tasks in feature extraction and classification:

1. **Convolutional Layer**: This is where the network learns the features of an image by applying filters (also known as kernels) to the input image.
2. **Activation Function (ReLU)**: ReLU (Rectified Linear Unit) is applied after each convolutional layer to introduce non-linearity, allowing the model to learn more complex patterns.
3. **Pooling Layer**: This layer reduces the dimensionality of the data, typically by using MaxPooling, which picks the maximum value from each patch of the feature map, helping the model to become invariant to small translations of the input image.
4. **Dropout**: This layer is used during training to randomly disable some neurons, helping the model avoid overfitting.
5. **Fully Connected Layer (Dense Layer)**: This layer connects every neuron from the previous layer to every neuron in the current layer. The final fully connected layer outputs the probabilities of each class (in this case, the 7 facial expression categories).

The key advantage of CNNs is their ability to automatically learn relevant features from raw images without requiring manual feature extraction, unlike traditional methods.

Working of CNN for Facial Expression Recognition

Let’s break down the algorithm step by step, using an example to explain it in simpler terms.

Step 1: Input Image

We start with an image of a person showing a facial expression. For example, consider an image where a person is smiling (happy expression).

The image is usually of size 48x48 pixels, and it is converted to grayscale, meaning it only contains shades of gray (no color), which simplifies the model's task. The grayscale image is then normalized by dividing each pixel value by 255, which scales the values to the range [0, 1].

Step 2: Convolutional Layer

Now, the CNN applies several filters (or kernels) to the image. These filters are small matrices (e.g., 3x3 or 5x5) that slide over the image to detect specific features like edges, textures, or shapes.

For instance:

- A filter may look for vertical edges in the image, which would help the model identify the outlines of the person's eyes or mouth.
- Another filter may detect horizontal edges, which could highlight the shape of the eyebrows.

After applying these filters, the CNN produces multiple feature maps, each representing a specific feature of the image.

Step 3: Activation Function (ReLU)

Once the convolutional layer applies the filters, an activation function, typically **ReLU (Rectified Linear Unit)**, is applied. The ReLU function replaces all negative values with zero and keeps positive values unchanged. This step introduces non-linearity into the model, allowing it to learn more complex patterns in the data.

For example, after applying the filters and ReLU activation, we might get a feature map that highlights the edges of the eyes and mouth for a "happy" expression.

Step 4: Pooling Layer

Next, a **MaxPooling** layer is used to reduce the size of the feature maps while preserving the most important information. In MaxPooling, the image is divided into small patches (e.g., 2x2), and only the maximum value from each patch is kept.

This reduces the computational cost and prevents overfitting by making the model invariant to small changes in the image, such as slight rotations or translations of the face.

For example, after MaxPooling, the model may reduce the size of the feature maps while still keeping the essential information about the eyes and mouth.

Step 5: Dropout Layer

To prevent overfitting (where the model learns too much from the training data and performs poorly on new data), the model randomly "drops out" a portion of the neurons during training. This means that some neurons are ignored in each iteration, forcing the model to rely on a broader set of features.

Step 6: Fully Connected Layer (Dense Layer)

After the convolution and pooling layers, the extracted features are flattened into a 1D vector and passed through one or more fully connected layers. The fully connected layers are used to classify the image into one of the predefined categories (in this case, seven emotions: angry, disgust, fear, happy, neutral, sad, and surprise).

For example, after processing the features of the happy face, the model will output a vector of probabilities, one for each emotion category. The model will predict the emotion with the highest probability.

Step 7: Output Layer

The final output layer consists of 7 neurons (one for each emotion), and the model assigns a probability to each of them. The output with the highest probability is chosen as the predicted label.

For example, if the model outputs the following probabilities:

- Angry: 0.01
- Disgust: 0.05
- Fear: 0.03
- Happy: 0.85
- Neutral: 0.02
- Sad: 0.02
- Surprise: 0.02

The model would predict the "happy" label because it has the highest probability (0.85).

Example: Predicting Emotion from an Image

Let’s go through a complete example where we predict the emotion of a person from a sample image of their face:

1. **Input**: An image of a person smiling (happy expression) is taken as input.
2. **Preprocessing**: The image is resized to 48x48 pixels and converted to grayscale.
3. **Feature Extraction**: The CNN applies filters to detect features like edges of the eyes, eyebrows, and mouth. These features help the model understand the facial structure and expressions.
4. **Classification**: The features are passed through several layers of convolution, pooling, and dropout to ensure they are robust and generalized. The output is a probability distribution across the seven emotion categories.
5. **Prediction**: The model outputs the probabilities for each emotion. The highest probability corresponds to the predicted emotion, which in this case would be "happy."

Final Prediction

The model predicts that the person in the image is "happy" based on the features it learned and classified using CNN.

The model architecture consists of several convolutional layers followed by pooling layers and dropout for regularization. The output is passed through a fully connected dense layer.

Building the CNN model

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense

model = Sequential()
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(7, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=128, epochs=100, validation_data=(x_test, y_test))
```

The CNN (Convolutional Neural Network) model is constructed to effectively identify emotions from images. The architecture starts with the Sequential model from Keras, which allows for stacking layers sequentially.

The first layer is a Conv2D layer with 128 filters, a 3x3 kernel size, and ReLU activation. This layer extracts features from the input image, such as edges and textures, which are crucial for emotion recognition. The input shape is set to (48, 48, 1), representing 48x48 grayscale images.

The MaxPooling2D layer follows, which reduces the spatial dimensions of the feature maps by half, helping to lower computational cost and make the model more robust to small shifts in the input.

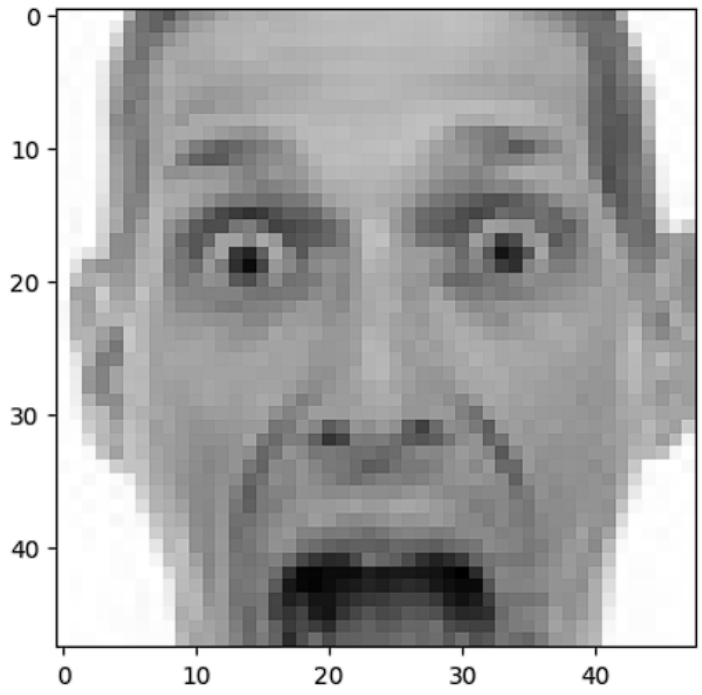
A Dropout layer with a rate of 0.4 is added to prevent overfitting by randomly dropping 40% of the neurons during training.

The Flatten layer converts the 2D feature maps into a 1D vector to feed into the fully connected layers. The final Dense layer has 7 output neurons, corresponding to the seven emotion classes, and uses the softmax activation function to output probability distributions over the classes.

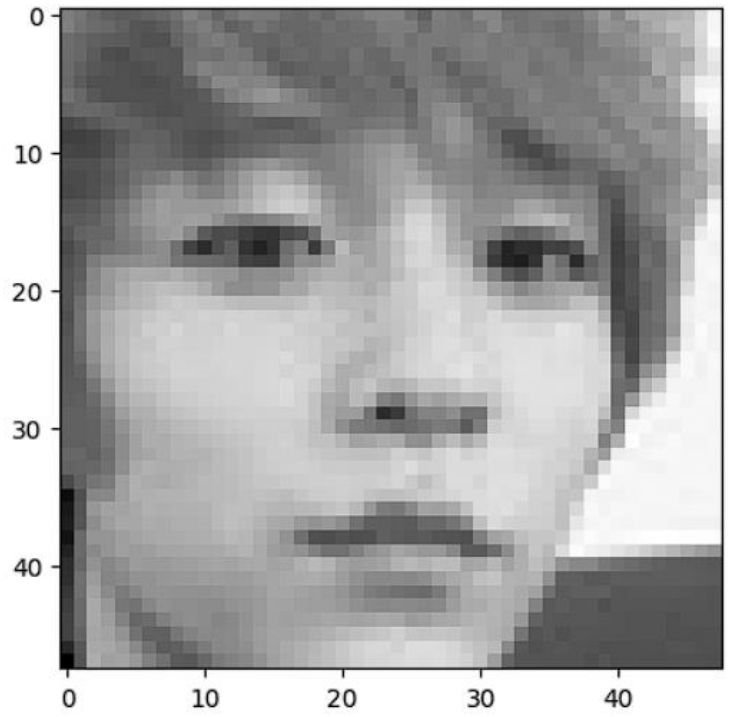
The model is compiled using the Adam optimizer, which adapts the learning rate for efficient training, and the categorical_crossentropy loss function, suitable for multi-class classification tasks. The fit() method trains the model over 100 epochs with a batch size of 128, and validation_data is used to evaluate performance on unseen data during training.

Some output images

```
original image is of fear
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
model prediction is  fear
<matplotlib.image.AxesImage at 0x1ad706135e0>
```




```
original image is of neutral
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
model prediction is  neutral
<matplotlib.image.AxesImage at 0x1ad706e7280>
```



Step 5: Real-Time Emotion Detection

Using OpenCV, the trained model is applied to video streams for real-time emotion detection.

Real-Time Emotion Detection

```
import cv2
webcam = cv2.VideoCapture(0)
labels = {0: 'angry', 1: 'disgust', 2: 'fear', 3: 'happy', 4: 'neutral', 5: 'sad', 6:
'surprise'}

while True:
    _, im = webcam.read()
    gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(im, 1.3, 5)
    for (p, q, r, s) in faces:
        image = gray[q:q+s, p:p+r]
        image = cv2.resize(image, (48, 48))
        img = np.array(image).reshape(1, 48, 48, 1) / 255.0
        pred = model.predict(img)
        prediction_label = labels[pred.argmax()]
        cv2.putText(im, prediction_label, (p-10, q-10), cv2.FONT_HERSHEY_COMPLEX_SMALL, 2, (0,
0, 255))
    cv2.imshow("Output", im)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
webcam.release()
cv2.destroyAllWindows()
```

For real-time emotion detection, OpenCV is used to capture and process video streams from a webcam. The process begins with starting a video capture using cv2.VideoCapture(0), which accesses the default camera. The video frames are read in a loop and converted to grayscale using cv2.cvtColor() for easier face detection.

The face_cascade.detectMultiScale() method is employed to identify faces within each frame. Detected faces are cropped and resized to 48x48 pixels to match the input size expected by the model. These images are normalized and reshaped to the format (1, 48, 48, 1) before being fed into the pre-trained model.

The model's prediction is obtained using model.predict(), which outputs the probabilities for each emotion class. The label with the highest probability is chosen using pred.argmax(), and the corresponding emotion is displayed on the video frame using cv2.putText(). The annotated frame is shown in a window with cv2.imshow().

The loop continues until the 'q' key is pressed, at which point the video capture is released, and all OpenCV windows are closed using webcam.release() and cv2.destroyAllWindows(). This setup enables continuous real-time emotion recognition from the video feed.

8. Results and Discussion

The model's performance, reflected in a training accuracy of 71.03% and a validation accuracy of 62.72%, indicates that while it performs reasonably well, there are factors limiting its effectiveness. While the accuracy is decent, there is room for improvement. This discrepancy between training and validation accuracy suggests potential overfitting, where the model learns the training data well but struggles to generalize to unseen data.

Several factors could contribute to this:

- **Insufficient Training Data or Imbalance**: A limited amount of training data or a dataset that is imbalanced, where certain emotion categories are underrepresented, can negatively affect the model's ability to learn comprehensive features across all emotions. This may lead to biased predictions favoring more common emotions while failing to recognize rarer ones effectively.

- **Variations in Lighting and Face Orientations**: In real-world scenarios, facial expressions are captured under different lighting conditions and from various angles. If the training data does not adequately represent these variations, the model may struggle to perform well during testing or real-time applications where such inconsistencies are present.

- **Model Complexity**: The model architecture, while effective, is relatively simple and might not capture all the complex features required for emotion detection. Enhancing the model with additional convolutional layers, batch normalization, or more sophisticated architectures like transfer learning using pre-trained models could improve its generalization ability.

- **Preprocessing Limitations**: Although images are normalized and resized, additional preprocessing techniques like data augmentation (flipping, rotating, zooming) could be employed to make the model more resilient to different facial orientations and backgrounds.

9. Conclusion

The project showcased the potential of convolutional neural networks (CNNs) for real-time emotion recognition, proving that even a relatively simple architecture can yield meaningful results. The system successfully identifies emotions such as anger, happiness, and surprise by processing video feed inputs through pre-trained deep learning models.

However, the results highlighted several avenues for potential enhancement. In addition to improving model accuracy and robustness, future work could focus on refining the data preprocessing steps, such as applying more extensive data augmentation to handle variability in lighting, facial orientations, and backgrounds. Augmentation can help the model generalize better to real-world scenarios.

Exploring alternative and more sophisticated model architectures, such as **ResNet** or **VGGNet**, which have deeper structures and proven efficacy in feature extraction, could substantially boost performance. Transfer learning from pre-trained models on large datasets could be particularly effective, reducing the need for vast amounts of labeled data and speeding up the training process.

Moreover, enhancing the model's regularization techniques and experimenting with hyperparameter tuning could help mitigate overfitting and improve generalization. Integrating techniques like **batch normalization** and **learning rate schedules** could also lead to performance gains.

Finally, expanding the dataset to include diverse facial expressions across different age groups, ethnicities, and environments would further strengthen the model's capability. Implementing real-world tests and feedback loops can make the system adaptive and more resilient to variations in user behavior and environmental conditions.

This project lays the groundwork for scalable, real-time emotion detection systems that could find applications in user experience improvement, psychological studies, and human-computer interaction enhancements.

10. Applications

1. **Healthcare and Psychology**: Identifying emotions can assist in monitoring mental health and detecting stress or depression.
2. **Human-Computer Interaction (HCI)**: Emotion recognition can enhance interactions with robots and virtual agents, making them more responsive to user emotions.
3. **Surveillance and Security**: Facial expression analysis can be used in surveillance systems to detect suspicious behavior based on emotional cues.
4. **Marketing and Customer Feedback**: Emotion detection can help companies assess customer satisfaction levels in real-time, such as during product demos or at retail checkouts.

11. Limitations

- **Dataset Bias**: The model's performance may be limited by the size and diversity of the training dataset.
- **Lighting and Angle Variations**: The model may not perform as well under extreme lighting conditions or large facial angle variations.
- **Occlusions**: Objects covering parts of the face (e.g., masks) can hinder the model's ability to correctly classify emotions.

12. References

1. K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," ICLR, 2015.
2. M. Bartlett, J. Cohn, and H. Lee, "Facial Expression Recognition: A Survey," 2001.
3. A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," NeurIPS, 2012.