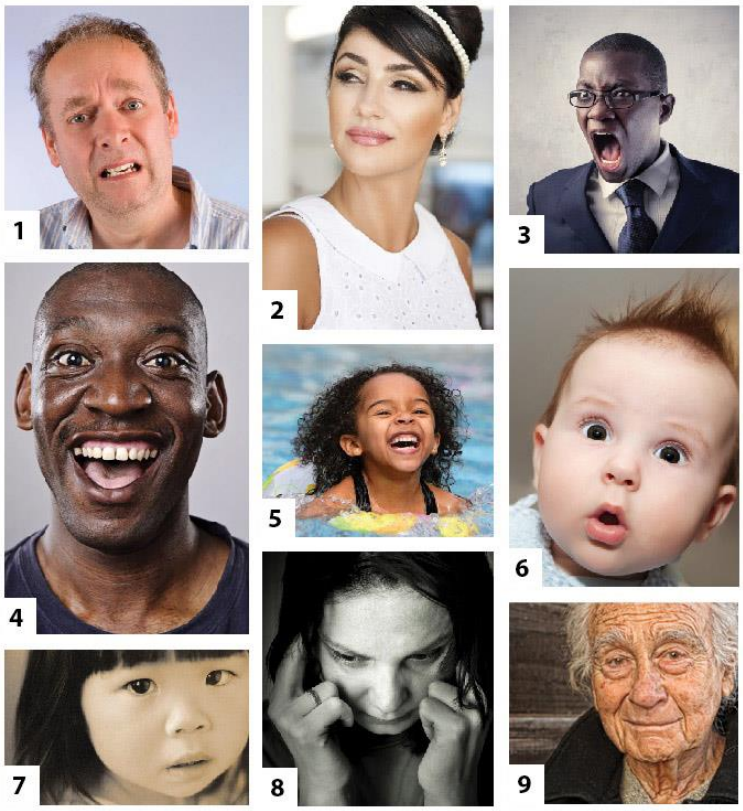

EMOTION DETETCTION

REAL-TIME VIDEO

DHARANYA KANAGARAJ | 06-06-2024

Emotion Detection Project

Introduction

The Emotion Detection project, is an application of artificial intelligence and computer vision that identifies and interprets human emotions from facial expressions in real-time videos using Convolutional Neural Networks (CNNs). The project involves several steps including data collection, data preparation, image augmentation, and model building and training. A simplified Python code snippet for training a CNN model for emotion detection is provided. The project also includes the creation of a GUI for video capture and an explanation of the code used for real-time video capture and emotion prediction. This project demonstrates the potential of deep learning and computer vision for real-time emotion detection and has numerous practical applications.

Emotion Detection Using Convolutional Neural Networks (CNNs)

Emotion detection, also known as facial emotion recognition, is a field within artificial intelligence and computer vision. It involves identifying and interpreting human emotions from facial expressions. Here's a brief overview of how it works with CNNs:

1. **Understanding Emotion Detection**: Emotions are expressed through facial expressions, body language, and voice tone. Among these, facial expressions are often the most visible and reliable indicators of emotion.
2. **Convolutional Neural Network (CNN) Architecture**: CNNs are a type of deep learning neural network architecture designed for processing grid-like data, such as images and videos. They are particularly effective at capturing spatial hierarchies of features in data.
3. **Emotion Detection Process**: Emotion detection using CNNs primarily focuses on analyzing facial expressions to determine the emotional state of an individual. The process typically involves the following steps:
    - **Data Collection**: A dataset containing labeled facial expressions is collected. Each image in the dataset is labeled with the corresponding emotion (e.g., happy, sad, angry).
    - **Preparing Data**: The data is pre-processed to be in the right format.
    - **Image Augmentation**: This step involves creating modified versions of images in the dataset to reduce overfitting and improve the model's ability to generalize.
    - **Model Building and Training**: A CNN model is built and trained using the prepared dataset.
4. **Applications**: Accurate emotion detection has numerous practical applications, including human-computer interaction, customer feedback analysis, mental health monitoring, driver's drowsiness detection, and students' behavior detection.

In conclusion, CNNs have revolutionized the field of emotion detection, significantly improving our ability to understand and process emotional cues from images.

Simplified base structure of model designing and training:

Here is a simplified Python code snippet for training a Convolutional Neural Network (CNN) model for emotion detection:

```
# Import necessary libraries

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense


# Initialize the model

model = Sequential()


# Add layers

model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(units=128, activation='relu'))

model.add(Dense(units=7, activation='softmax'))  # 7 units for 7 emotions


# Compile the model

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])


# Load your dataset

# X_train, y_train = load_your_data()


# Train the model

# model.fit(X_train, y_train, epochs=10, batch_size=32)
```

This code initializes a simple CNN model with one convolutional layer, one max pooling layer, and two fully connected layers. The model is then compiled with the Adam optimizer and the categorical cross-entropy loss function, which are commonly used for multi-class classification problems[1].

The X_train and y_train variables should contain the training data and corresponding labels, respectively.

This is a very basic model for illustrative purposes. In practice, we have pre-processing steps, and techniques to prevent overfitting, such as dropout or regularization.

Creation GUI for video capture:

This project is an implementation of a real-time emotion detection system using a pre-trained model, OpenCV, and Keras. The Python script provided performs emotion detection on a video file. Here's a detailed breakdown of the project:

1. **Importing Libraries**: The script begins by importing the necessary libraries. cv2 is used for image and video processing, numpy for numerical operations, and keras.models for loading the trained model.
2. **Emotion Dictionary**: An emotion dictionary is defined to map the output of the model to the corresponding emotion. This dictionary is used later to display the predicted emotion on the video frame.
3. **Loading the Model**: The script loads a pre-trained model from a JSON file and its corresponding weights from a .h5 file. This model is used to predict the emotion from the facial expressions in the video frames.
4. **Video Capture**: The script captures video from a specified file. It can be modified to use a webcam feed by uncommenting the appropriate line.
5. **Face Detection and Emotion Prediction**: In an infinite loop, the script reads frames from the video, resizes them, and converts them to grayscale. It then uses a Haar cascade classifier to detect faces in the frame. For each detected face, the script extracts the region of interest (ROI), resizes it to the input size expected by the model, and expands its dimensions to match the model's input shape. The model then predicts the emotion for the ROI, and the script draws a rectangle around the face and writes the predicted emotion above the rectangle.
6. **Display and Termination**: The script displays the processed frame with the detected faces and their predicted emotions. The loop continues until the 'q' key is pressed, at which point the script releases the video capture and closes all OpenCV windows.

This project demonstrates the application of deep learning and computer vision for real-time emotion detection. It can be extended or modified for various applications such as emotion-based recommendation systems, psychological studies, interactive games, and more. The project could benefit from further enhancements such as improving the accuracy of the emotion detection model, supporting multiple video formats, and optimizing the script for better performance.

Explanation of the code

Here's a breakdown of the code:

```
import cv2
import numpy as np
from keras.models import model_from_json
```

The script begins by importing the necessary libraries. `cv2` is used for image and video processing, `numpy` for numerical operations, and `keras.models` for loading the trained model.

```
emotion_dict={0:'Angry',                    1:'Disgusted',
2:'Fearful',3:'Happy',4:'Neutral',5:'Sad',6:'Suprised'}
```

An emotion dictionary is defined to map the output of the model to the corresponding emotion. This dictionary is used later to display the predicted emotion on the video frame.

```
json_file=open(r'C:\Users\admin\Documents\data
science\Projects\Emotion
Detection\Video&images\model_a1.json','r')
loaded_model_json = json_file.read()
json_file.close()
emotion_model = model_from_json(loaded_model_json)
```

The script loads a pre-trained model from a JSON file. This model is used to predict the emotion from the facial expressions in the video frames.

```
emotion_model.load_weights(r'C:\Users\admin\Documents\data
science\Projects\Emotion
Detection\Video&images\model1.weights.h5')
print('loaded model from disk')
```

The script then loads the weights of the model from a `.h5` file.

```
cap=cv2.VideoCapture(r"C:\Users\admin\Documents\data
science\Projects\Emotion       Detection\Video&images\Sample
videos\emotion_video_sample1.mp4")
```

The script captures video from a specified file. It can be modified to use a webcam feed by uncommenting the appropriate line.

```
while True:
    ret,frame = cap.read()
    frame = cv2.resize(frame,(1280,720))
    if not ret:
        print("Error: Failed to capture frame.")
        break
    face_cascade_path     =     cv2.data.haarcascades     +
'haarcascade_frontalface_default.xml'
    face_cascade = cv2.CascadeClassifier(face_cascade_path)
    gray_frame = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
```

In an infinite loop, the script reads frames from the video, resizes them, and converts them to grayscale. It then uses a Haar cascade classifier to detect faces in the frame.

```
num_faces     =     face_cascade.detectMultiScale(gray_frame,
scaleFactor = 1.3, minNeighbors = 5)
```

The script detects faces in the grayscale frame.

```
for (x,y,w,h) in num_faces:
    cv2.rectangle(frame,(x,y-50),(x+w,y+h+10),(0,255,0),4)
    roi_gray_frame = gray_frame[y:y + h,x:x + w]
    cropped_img                                                =
np.expand_dims(np.expand_dims(cv2.resize(roi_gray_frame,(48
,48)),-1),0)
    emotion_prediction = emotion_model.predict(cropped_img)
    maxindex = int(np.argmax(emotion_prediction))
    cv2.putText(frame, emotion_dict[maxindex], (x+5, y-20),
cv2.FONT_HERSHEY_SIMPLEX,1, (255,0,0),2,cv2.LINE_8)
```

For each detected face, the script extracts the region of interest (ROI), resizes it to the input size expected by the model, and expands its dimensions to match the model's input shape. The model then predicts the emotion for the ROI, and the script draws a rectangle around the face and writes the predicted emotion above the rectangle.

```
cv2.imshow('Emotion Detection', frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

The script displays the processed frame with the detected faces and their predicted emotions. The loop continues until the ‘q’ key is pressed, at which point the script releases the video capture and closes all OpenCV windows.

```
cap.release()
cv2.destroyAllWindows()
```

The script releases the video capture and closes all OpenCV windows. This is done when the ‘q’ key is pressed, breaking the loop and ending the program.

Conclusion

The Emotion Detection project successfully demonstrates the application of Convolutional Neural Networks (CNNs) in the field of artificial intelligence and computer vision for real-time emotion detection. The project effectively utilizes CNNs to analyze facial expressions and determine the emotional state of an individual.

The project includes a simplified Python code snippet for training a CNN model for emotion detection, and a detailed explanation of the code used for real-time video capture and emotion prediction. The project also highlights the practical applications of accurate emotion detection, such as in human-computer interaction, customer feedback analysis, mental health monitoring, driver's drowsiness detection, and students' behavior detection.

The project could be further enhanced by improving the accuracy of the emotion detection model, supporting multiple video formats, and optimizing the script for better performance. It can also be extended or modified for various applications such as emotion-based recommendation systems, psychological studies, interactive games, and more.

In conclusion, this project significantly contributes to the field of emotion detection, demonstrating the potential of CNNs in understanding and processing emotional cues from images and real-time videos. The project serves as a valuable resource for anyone interested in the practical implementation of CNNs for emotion detection.