Algorithm for Custom CNN Model

Import necessary modules from keras for model creation and image preprocessing.

Import os, pandas, numpy, tqdm for data handling, LabelEncoder from sklearn.preprocessing.

Set TRAIN_DIR and TEST_DIR to paths containing the training and testing image datasets.

Define createdataframe function

Function createdataframe(dir):

> Initialize image_paths and labels as empty lists.
>
> For each label in the given directory:
>
> > For each image in the label's subdirectory:
> >
> > > Append the image path to image_paths.
> > >
> > > Append the label to labels.
> >
> > Print completion message for the current label.
>
> Return image_paths and labels.

Define extract_features function

Function extract_features(images):

> Initialize features as an empty list.
>
> For each image in the list:
>
> > Load the image in grayscale mode.
> >
> > Convert the image into a NumPy array.
> >
> > Append the array to features.
>
> Convert features to a NumPy array.
>
> Reshape the array to (n_samples, 48, 48, 1).
>
> Return the reshaped array.

train['image'], train['label'] = createdataframe(TRAIN_DIR)

test['image'], test['label'] = createdataframe(TEST_DIR)

train_features = extract_features(train['image'])

test_features = extract_features(test['image'])

x_train = train_features / 255.0

x_test = test_features / 255.0

Initialize a LabelEncoder.

Fit the encoder with train['label'].

y_train = le.transform(train['label'])

y_test = le.transform(test['label'])

Convert y_train and y_test into one-hot encoded format using to_categorical().

Initialize a Sequential model.

Add convolutional layers:

> Add Conv2D with 128 filters, kernel size (3,3), and activation 'relu'.
>
> Add MaxPooling2D with pool size (2,2).
>
> Add a Dropout layer with a rate of 0.4.
>
> Repeat the above steps for layers with 256, 512, and 512 filters.
>
> Add a Flatten layer.

Add fully connected layers:

> Add a Dense layer with 512 neurons and activation 'relu'.
>
> Add a Dropout layer with a rate of 0.4.
>
> Add another Dense layer with 256 neurons and activation 'relu'.
>
> Add a Dropout layer with a rate of 0.3.

Add an output layer:

> Add a Dense layer with 7 neurons and activation 'softmax'.

Use 'adam' as the optimizer.

Set the loss function to 'categorical_crossentropy'.

Specify accuracy as the evaluation metric.

Call model.fit() with the following parameters:

> x_train, y_train as input.
>
> Batch size: 128.
>
> Epochs: 100.
>
> Validation data: (x_test, y_test).

Save the model architecture to a JSON file:

> Write model.to_json() to emotiondetector.json.

Save the model weights to an H5 file:

> model.save("emotiondetector.h5").

Algorithm for Emotion Detection and Scoring Using a CNN Model

Import necessary modules: sys, io, cv2, time, and others for handling model, webcam, and data processing.

Redirect sys.stdout to support UTF-8 encoding.

Define json_file_path and weights_file_path for loading the model architecture and weights.

Load the pre-trained model

Try to:

> Open and read the JSON file containing the model architecture.
>
> Load the model using model_from_json().
>
> Load the model weights from the specified file path.

Catch FileNotFoundError and print an error message if the files are not found, then exit.

Set haar_file to the path of the Haar Cascade XML file for frontal face detection.

Initialize the face_cascade classifier with the Haar file.

Define extract_features function

Function extract_features(image):

> Convert the given image to a NumPy array.
>
> Reshape the array to (1, 48, 48, 1).
>
> Normalize the values by dividing by 255.0.
>
> Return the processed array.

Open the webcam using cv2.VideoCapture(0).

Check if the webcam opens successfully; if not, print an error message and exit.

Set a dictionary labels mapping numerical indices to emotion names.

Initialize variables

> detected_emotions: An empty list to store detected emotions.
>
> start_time: The current time to track the elapsed time.

Start video capture loop

While True:

> Calculate elapsed_time as the difference between the current time and start_time.
>
> Break the loop if elapsed_time exceeds 30 seconds.
>
> Capture an image frame from the webcam.
>
> Convert the captured image to grayscale.

> Detect faces using the Haar Cascade classifier.
>
> For each detected face:
>
> > Extract the region of interest (ROI) from the grayscale image.
> >
> > Draw a rectangle around the detected face in the original frame.
> >
> > Resize the ROI to `(48, 48)`.
> >
> > Extract features using `extract_features()`.
> >
> > Try to:
> >
> > > Predict the emotion using the loaded model.
> > >
> > > Map the prediction to a label using `labels`.
> > >
> > > Append the predicted emotion to `detected_emotions`.
> > >
> > > Display the emotion label on the video frame.
> >
> > Catch exceptions during prediction or display and print the error message.
>
> Display the video frame with bounding boxes and emotion labels.
>
> Break the loop if the 'q' key is pressed.

Release the webcam.

Close all OpenCV windows.

Define emotion scores and counts

> Set emo_score_1: A dictionary mapping each emotion to a score.
>
> Set emo_count: A dictionary to count occurrences of each emotion (initialized to 0).

Use reduce() to compute total_score as the sum of scores for all detected emotions based on emo_score_1.

Print:

> The list of detected_emotions over 30 seconds.
>
> The number of detected emotions.
>
> The total_score rounded to 3 decimal places.
>
> The average score (total divided by the number of emotions), rounded to 3 decimal places.

Algorithm for Depression Score Calculation

Configure page with title, icon, layout, and sidebar state.

Apply custom CSS for styles.

Display:

> Title: "Emotion Echo"
>
> Subtitle: "Helping you understand and manage your emotions."

Define questions as:

> List of dictionaries, each containing:
>
> > Question Text
> >
> > Options: (Answer, Score)

Define `assess_depression(total_score)`:

> If score ≤ 9: return "Minimal Depression"
>
> Else if score 10–18: return "Mild Depression"
>
> Else if score 19–29: return "Moderate Depression"
>
> Else: return "Severe Depression"

Initialize `total_score` to 0.

Create a form for questions:

> For each question in the list:
>
> > Display question text.
> >
> > Use radio buttons for answer selection.
> >
> > Add the score of the selected answer to `total_score`.

Add a submit button.

If form is submitted:

> Calculate `depression_level` using `total_score`.
>
> Display:
>
> > Total Inventory Score: {total_score}
> >
> > Depression Level: {depression_level}

Display footer with message: "Emotion Echo | Bringing positivity to your life."

Algorithm for Calculating Anxiety Score

Input values:

> Emotion Score (denoted as emotion_score)
>
> Heart Rate (denoted as heart_rate)
>
> Depression Score (denoted as depression_level)

Coefficients:

> α (alpha): Weight for Emotion Score
>
> β (beta): Weight for Heart Rate Score
>
> γ (gamma): Weight for Depression Level

Normalise the values:

> Compute heart_rate_score as (heart_rate - 60) / 100.
>
> Compute depression_score as depression_level / 30.

Apply the formula:

> anxiety_score = (α × emotion_score) + (β × (heart_rate_score) + (γ × depression_score)

Display the anxiety_score.