Algorithm for Custom CNN Model

1. **Import Required Libraries**
    - Import necessary modules such as Keras for building the model, and libraries for data preprocessing and handling (e.g., NumPy, Pandas).
2. **Load and Preprocess Data**
    - Load the dataset and preprocess it as follows:
        - Resize images to a uniform shape.
        - Normalize pixel values to a range of [0, 1].
        - Convert labels to one-hot encoded format using to_categorical.
3. **Initialize the Model**
    - Create a Sequential model using Sequential().
4. **Add Convolutional and Pooling Layers**
    - Add a Conv2D layer with 128 filters, a (3x3) kernel size, and ReLU activation. Set the input shape for the first layer.
    - Add a MaxPooling2D layer with a (2x2) pool size.
    - Add a Dropout layer with a dropout rate of 0.4 to prevent overfitting.
    - Repeat similar steps for additional Conv2D layers with increasing filters (256, 512) as needed, followed by MaxPooling2D and Dropout layers.
5. **Flatten the Output**
    - Add a Flatten layer to convert the 2D output from the convolutional layers into a 1D vector.
6. **Add Fully Connected Layers**
    - Add a Dense layer with 512 units and ReLU activation.
    - Add a Dropout layer with a dropout rate of 0.4.
    - Add another Dense layer with 256 units and ReLU activation.
    - Add a Dropout layer with a dropout rate of 0.3.
7. **Add the Output Layer**
    - Add a Dense layer with 7 units (corresponding to the number of classes) and softmax activation for multiclass classification.
8. **Compile the Model**
    - Compile the model using the following parameters:
        - Optimizer: Adam
        - Loss function: Categorical Crossentropy

        - Metrics: Accuracy

9. **Train the Model**
    - Train the model using the fit method:
        - Inputs: Training data (x_train, y_train).
        - Batch size: 128.
        - Number of epochs: 100.
        - Validation data: (x_test, y_test).

10. **Save the Model**
    - Save the model architecture to a JSON file using model.to_json().
    - Save the model weights to an H5 file using model.save().

Algorithm for Emotion Detection and Scoring Using a CNN Model

1. **Import Required Libraries**
   - Import libraries such as sys, io, cv2 (OpenCV), time, and Keras modules (model_from_json).
   - Import additional utilities like NumPy and reduce from functools.
2. **Set Up Default Encoding**
   - Set the standard output encoding to UTF-8 for compatibility.
3. **Load the Pre-trained CNN Model**
   - Specify file paths for the model architecture (emotiondetector.json) and weights (emotiondetector.h5).
   - Load the model architecture from the JSON file using model_from_json().
   - Load the pre-trained weights into the model.
4. **Load Haar Cascade Classifier**
   - Load the Haar Cascade XML file for face detection.
5. **Define Feature Extraction Function**
   - Create a function extract_features() to preprocess input images:
     - Reshape the image to (1, 48, 48, 1).
     - Normalize pixel values to the range [0, 1].
6. **Initialize Webcam**
   - Open the webcam using cv2.VideoCapture(0).
   - Check if the webcam is accessible. Exit if not.
7. **Initialize Variables**
   - Define a dictionary of emotion labels corresponding to model output indices.
   - Create a list to store detected emotions.
   - Start a timer to limit the program runtime to 30 seconds.
8. **Start Emotion Detection Loop**
   - Loop continuously while the elapsed time is under 30 seconds:
     1. Capture frames from the webcam.
     2. Convert the frame to grayscale using cv2.cvtColor().
     3. Detect faces using the Haar Cascade.
     4. For each detected face:
        - Extract the face region.

- Resize the face to 48x48 pixels.
- Preprocess the face using extract_features().
- Use the CNN model to predict the emotion.
- Append the detected emotion to the list.
- Draw a rectangle around the face and label it with the predicted emotion.

5. Display the output frame with labeled emotions.
6. Exit the loop if the 'q' key is pressed.

9. **Release Resources**
   - Release the webcam and close all OpenCV windows.
10. **Define Emotion Scoring Dictionaries**
    - Define dictionaries (emo_score_1 and emo_score_2) to assign scores to each emotion.
    - Initialize an emotion count dictionary to keep track of occurrences.
11. **Calculate Emotion Scores**
    - Compute the total emotion score using reduce() and the emo_score_1 dictionary.
    - Print the list of detected emotions, their count, total score, and average score.
12. **Exclude Neutral Emotions and Recompute Scores**
    - Filter out "neutral" emotions from the detected list.
    - Compute the total score and average score for the filtered list.
    - Print the filtered emotions, count, total score, and average score.

Algorithm for Depression Score Calculation

1. **Import Required Libraries**
   - Import streamlit for creating the user interface.
2. **Set Up Page Configuration**
   - Configure the Streamlit app using st.set_page_config() with the following parameters:
     - Page title: "Emotion Echo - Depression Assessment"
     - Page icon: "🌅"
     - Layout: Centered layout
3. **Customize Page Style**
   - Use HTML and CSS to:
     - Set the background color to dark mode.
     - Customize font styles, title, and button colors.
4. **Display Title and Subtitle**
   - Use HTML tags to center-align and style the app title "Emotion Echo" and subtitle "Helping you understand and manage your emotions."
5. **Define Questions and Options**
   - Create a list of dictionaries where each dictionary contains:
     - Question text (e.g., "Mood")
     - Options with corresponding scores.
   - Each question is designed to assess depression factors such as mood, pessimism, self-hate, irritability, etc.
6. **Create a Depression Assessment Form**
   - Use st.form() to define an interactive form with:
     - Each question displayed using st.markdown().
     - Answer options displayed as radio buttons (st.radio()).
     - Calculate the depression score by adding the corresponding scores based on the user's response.
7. **Define Depression Assessment Logic**
   - Create a function assess_depression() to categorize total scores into depression levels:
     - 0-9: Minimal Depression
     - 10-18: Mild Depression

    - 19-29: Moderate Depression
    - 30 and above: Severe Depression

8. **Display Assessment Results**
   - On form submission, calculate the total score and display:
     - Total Inventory Score
     - Depression Level (determined by the assess_depression() function).

9. **Add a Footer**
   - Include a footer section styled using HTML, containing the app name and tagline: "Emotion Echo | Bringing positivity to your life 🌅."

Algorithm for Calculating Anxiety Score

1. **Inputs**:
   - Obtain the following inputs:
     - **Emotion Score**: A numerical value (e.g., from emotion detection).
     - **Heart Rate**: The heart rate of the individual (in beats per minute).
     - **Depression Score**: A numerical value representing the depression level.
     - Constants: α, β, and γ (weighting factors).
2. **Normalize the Inputs**:
   - Normalize the Heart Rate using the formula:
     - Normalized Heart Rate= [Heart Rate−60]/100
   - Normalize the Depression Score using the formula:
     - Normalized Depression Score=Depression Score/30
3. **Calculate Anxiety Score**:
   - Use the formula to calculate the Anxiety Score:
     - Anxiety Score = α × Emotion Score + β × Normalized Heart Rate + γ × Normalized Depression Score
4. **Output the Result**:
   - Return the calculated Anxiety Score.
5. **Validation (Optional)**:
   - Check if all input values are within valid ranges:
     - Emotion Score: Valid range depends on its source.
     - Heart Rate: Typically between 40 and 200 bpm.
     - Depression Score: Based on the assessment scale used.