***{ ‘’’Code’’’* Explaination** }

**This code is designed to extract features from audio signals stored in `.wav` files in three different folders (representing different operational states of the motor), calculate key features from those signals, and save them into a CSV file. Here's a detailed explanation of how the code works:**

**1. Defining Paths to Folders and File Output Directory**

**```**

***folders = {'C:\Users\sonar\Desktop\Assignments\sem7 project\Dataset\train\_cut\engine1\_good',***

***'C:\Users\sonar\Desktop\Assignments\sem7 project\Dataset\train\_cut\engine2\_broken',***

***'C:\Users\sonar\Desktop\Assignments\sem7 project\Dataset\train\_cut\engine3\_heavyload'};***

***folderPath = 'C:\Users\sonar\Desktop\Assignments\sem7 project\Code';***

**```**

**- Three folder paths (`engine1\_good`, `engine2\_broken`, `engine3\_heavyload`) are provided, representing different operational states of the motor.**

**- The `folderPath` specifies where the final CSV file containing the extracted features will be saved.**

**2. Sampling Frequency and FFT Window Length**

**```**

***Fs = 44100; % Define sampling frequency***

***fft\_window\_length = 1024; % Define the FFT window length***

***chunk\_size = 10; % Number of files to process in each iteration***

**```**

**- `Fs = 44100`: The sampling frequency is set to 44.1 kHz, which is the standard sampling rate for audio signals.**

**- `fft\_window\_length = 1024`: This sets the number of data points used in the Fast Fourier Transform (FFT), which converts the time-domain signal into its frequency components.**

**- `chunk\_size = 10`: Determines how many files are processed in each iteration, allowing you to break the task into smaller, memory-efficient chunks.**

**3. Initializations**

**```**

***features = [];***

***labels = {}; % Initialize as a cell array for string labels***

**```**

**- `features` is initialized as an empty array where the extracted features will be stored.**

**- `labels` is initialized as an empty cell array where the labels for each file (corresponding to the folder the file came from) will be stored.**

**4. Loop Through Folders**

**```**

***for i = 1:length(folders)***

***currentFolder = folders{i};***

***files = dir(fullfile(currentFolder, '\*.wav'));***

**```**

**- This loop goes through each folder in the `folders` array.**

**- For each folder, it lists all `.wav` files in that folder.**

**5. Processing Files in Chunks**

**```**

***num\_chunks = ceil(length(files) / chunk\_size);***

***for k = 1:num\_chunks***

***start\_idx = (k-1) \* chunk\_size + 1;***

***end\_idx = min(k \* chunk\_size, length(files));***

***chunk\_files = files(start\_idx:end\_idx);***

**```**

**- This ensures the files are processed in chunks of size `chunk\_size` (10 files per chunk).**

**- Each iteration of the loop processes a set of files from `start\_idx` to `end\_idx`.**

**6. Loop Through Each Audio File and Extract Features**

**```**

***for j = 1:length(chunk\_files)***

***[signal, Fs] = audioread(fullfile(currentFolder, chunk\_files(j).name));***

***signal = single(signal); % Convert the signal to single precision***

**```**

**- For each file in the current chunk, the audio signal is read using `audioread`.**

**- The signal is converted to single-precision floating-point format to save memory and improve computational efficiency.**

**7. FFT Window Length Handling**

**```**

***if length(signal) < fft\_window\_length***

***signal = [signal; zeros(fft\_window\_length - length(signal), 1)];***

***elseif length(signal) > fft\_window\_length***

***signal = signal(1:fft\_window\_length);***

***end***

**```**

**- If the signal is shorter than the `fft\_window\_length` (1024 samples), it is padded with zeros to match the length.**

**- If the signal is longer than 1024 samples, it is truncated to exactly 1024 samples.**

**8. FFT Computation and Frequency Vector**

**```**

***Y = abs(fft(signal, fft\_window\_length)); % Compute FFT***

***N = length(Y);***

***f = (0:N-1) \* (Fs/N); % Frequency vector***

**```**

**- The FFT is applied to convert the time-domain signal into the frequency domain.**

**- `Y` contains the magnitudes of the FFT, representing the strength of different frequency components.**

**- `f` is the frequency vector that associates the indices of the FFT with specific frequencies.**

**9. Feature Extraction**

**- RMS Value: The Root Mean Square of the signal.**

**```**

***rms\_value = rms(signal);***

**```**

**- Mean Value: The average of the signal.**

**```**

***mean\_value = mean(signal);***

**```**

**- Variance: The variance, which shows how much the signal deviates from the mean.**

**```**

***variance\_value = var(signal);***

**```**

**- Crest Factor: The ratio of the peak amplitude to the RMS value.**

**```**

***crest\_factor = max(abs(signal)) / rms\_value;***

**```**

**- Energy: The total energy in the signal, which is the sum of squared amplitudes.**

**```**

***energy = sum(signal.^2);***

**```**

**- Dominant Frequency: The frequency with the highest amplitude in the FFT.**

**```**

***[~, idx] = max(Y);***

***dominant\_freq = f(idx);***

**```**

**10. Appending Features and Labels**

**```**

***current\_features = [rms\_value, mean\_value, variance\_value, crest\_factor, energy, dominant\_freq];***

***features = [features; current\_features];***

***labels = [labels; {folders{i}}];***

**```**

**- The extracted features are stored in the `features` array as a row vector for each file.**

**- The corresponding label (indicating the folder the file came from) is stored in the `labels` array.**

**11. Converting Features and Labels into a Table**

**```**

***labels = string(labels);***

***variableNames = {'RMS', 'Mean', 'Variance', 'CrestFactor', 'Energy', 'DominantFreq'};***

***data = array2table(features, 'VariableNames', variableNames);***

***data.Label = labels;***

**```**

**- The features are converted into a table with appropriate column names.**

**- The labels are added as a new column in the table.**

**12. Saving the Table to a CSV File**

**```**

***csvFilePath = fullfile(folderPath, 'features.csv');***

***writetable(data, csvFilePath);***

***disp(['CSV file saved to: ', csvFilePath]);***

**```**

**- The final table of features and labels is saved as a CSV file in the specified `folderPath`.**

**- The path to the saved file is displayed as a confirmation.**

**Summary**

**This code extracts six features (RMS, Mean, Variance, Crest Factor, Energy, and Dominant Frequency) from each audio signal in the specified folders, processes them in memory-efficient chunks, and writes the results to a CSV file.**